NATIONAL INSTITUTE of STANDARDS and TECHNOLOGY/
NATIONAL COMPUTER SECURITY CENTER

12th NATIONAL COMPUTER SECURITY CONFERENCE

BALTIMORE CONVENTION CENTER
BALTIMORE, MD
10 - 13 OCTOBER 1989

PROCEEDINGS

[illegible]

Welcome!

The National Computer Security Center (NCSC) and the National Computer Systems Laboratory (NCSL) are pleased to welcome you to the Twelfth Annual National Computer Security Conference. We believe that the Conference will stimulate a vital and dynamic exchange of information and foster an understanding of emerging technologies.

The theme for this year's conference--Information Systems Security: Solutions for Today, Concepts for Tomorrow--reflects the growing interest in the broader information systems security issues facing the user community. At the heart of these issues are two items which will receive special emphasis this week--Education, Training and Awareness, and Ethics. We firmly believe that security awareness and responsibility are the cornerstone of any information security program. Both the Federal Government and private industry must work together to build on that foundation; we believe this conference will serve both government and industry well in our cooperative efforts to explore and to apply state-of-the-art technology to information system security.

To be successful in our effort to establish a firm information systems security base, we ask you to share the information you learn this week with other users, managers and administrators. Only by sharing the knowledge with others can we hope to build on an even stronger foundation in the future.

JAMES H. BURROWS
Director
National Computer Systems Laboratory

PATRICK R. GALLAGHER, JR
Director
National Computer Security Center

TABLE OF CONTENTS

TRACK A - RESEARCH & DEVELOPMENT

STATEMENT "A" per Cathy Pudwell
National Computer Security Center/S33
Fort Meade, MD
TELECON 3/15/90 VG

TRACK B - SYSTEMS

TRACK C - MANAGEMENT & ADMINISTRATION

TRACK D - EDUCATION & ETHICS

ALTERNATE PAPERS

EXECUTIVE SUMMARIES

TRACK A

GOING BEYOND TECHNOLOGY TO MEET THE CHALLENGES OF MULTILEVEL DATABASE SECURITY

Gary W. Smith

School of Information Technology and Engineering
George Mason University
Fairfax, VA 22030

Abstract. In its quest for technical solutions to multilevel database security problems, the database security community appears to have lost sight of other aspects of security. This paper asserts that one can, in many instances, meet the challenges facing the database security community (e.g. polyinstantiation, integrity, Trojan horses and covert storage channels) through good system design and the use of management controls and procedural security. A conceptual framework for data access control and a conceptual operational framework are proposed to provide the requisite control in a database environment. The operational framework includes the notion of a privileged domain for programs that are authorized to update corporate data, and a less-privileged domain for read-only programs. Possible solutions to database security challenges are illustrated within the conceptual frameworks.

1 Introduction

The computer security community has been energized for the past few years to develop technological solutions to multilevel security (MLS) requirements. The approach to multilevel operating systems is in the engineering stage of development with many of the fundamental mechanisms in place. Proposed approaches and solutions in the database security arena in many cases have evolved from those used for operating systems. Database security, however, poses different challenges based upon the finer granularity at which multilevel security must be invoked.

Several issues continue to be topics of discussion and debate in the database security community: providing (and defining) integrity, the need for polyinstantiation, and protection from Trojan horses and covert channels. The community is actively designing database systems that will provide multilevel security [5,6,9,10,13,16,17]. These efforts, and the dominate theme in the literature, appear to be entirely dependent on technology to provide the required security. But long before there were computers, we had management controls, principles of good system design, and procedural security.

The orientation towards technology and mechanisms seems to be to the exclusion of more basic approaches to security (management controls, good system design, and procedural security). We assert that the challenges facing the database security community can in many instances be solved (and in the other instances made manageable) by incorporating non-technical aspects of security along those technical mechanisms that either are now, or will soon be, available.

The remainder of the paper is organized as follows. Section 2 provides a brief description, from a database perspective, of the challenges cited above. Section 3 gives a proposed solution including management and design concepts, a conceptual framework for data access control, and a conceptual operational framework. In Section 4 the proposed solution of Section 3 is used to illustrate solutions to the problems described in Section 2. Finally, the conclusions are given in Section 5.

2. Database Security Challenges

As researchers developed approaches and solutions to MLS requirements several significant challenges have been encountered. Some challenges are well-known because they have received considerable attention in the literature and are subject to intense debates.

2.1 Well-Known Challenges

Integrity. The Clark-Wilson paper of the 1987 Oakland Conference [4] started an intense debate on the issue of "integrity." The recent Invitational Data Integrity Workshop, sponsored by the National Institute for Science and Technology (NIST - formally the National Bureau of Standards) reemphasized the notion that "integrity" means many different things to different researchers. Many of the problems involving the community's difficulty in dealing with integrity relate to two areas: the different implicit meanings of "integrity" and the background of the individuals--specifically, the application domain from which they come and (more importantly) which they use to illustrate and understand the problem and proposed solutions.

To enhance precise communication both the meaning of integrity and application domain used are defined. For the purpose of this paper integrity involves two notions **who** can change the state of the database and **what** states of the database are valid. Changing the state of the database includes both allowing only authorized modification (i.e., update) of existing data (e.g., changing an employee's salary in the database) and allowing only authorized creation/deletion of data (e.g., adding a new employee to the database, or deleting a file or relation from the database). A valid state of the database means only that the data entered meets some criteria for validity (e.g., the department code entered is a valid department code). (Note that "valid data" does not mean that the data is correct, e.g., that the valid department code entered correctly reflects the department to which the employee is assigned.)

For the purpose of this paper the application domain is that of "structured data" that one might find in a database using a relational database management system (DBMS) supporting business or even many command and control applications. Note that the application domain is NOT that of text (and spelling checkers) or software development libraries, or CAD/CAM or a host of other domains.

Polyinstantiation. All the reasons and examples given which have driven the community to propose polyinstantiation can be grouped into two fundamental areas: the first area relates to update problems and the second area involves the need for cover stories.

The Woods Hole Study [1] identified a significant challenge in database security--how to hide the existence of classified data in a database when users (or code executing on behalf of a user which may contain a Trojan horse) inadvertently (or intentionally) try to update (add or modify) the classified data which they are not allowed to read directly by the security policy. Subsequently, the SEAVIEW project [6] proposed the concept of "polyinstantiation," that is multiple instances of the same data entity differing only by their classification level. The solution then is that when a user tries to update (change a data element or add a record) data the user is not authorized to see, the "system" will perform the update and polyinstantiate the data (i.e., provide muitiple instances of the record which differ by classification). The reason for having to do this, of course, is the policy that the user (or code operating on user's behalf) must not be told that there is data in the database which he or she is not authorized to see. This requirement was initially stated in the Hinke-Schaefer study [11] and has since been considered a "de facto standard" requirement for the development of MLS database systems.

The second fundamental challenge that polyinstantiation is designed to solve is that of cover stories. There are times when the organization wants to provide disinformation to users at lower levels of classification. The classic example is a classified destination for a military aircraft--you can't hide the existence of the aircraft nor that it is going to fly somewhere. But you want to hide its classified destination. So one lets the Top Secret user know the real destination of the flight (e.g., Iran), yet the Secret and lower users will be told a cover story (e.g., the destination is Greece). Implicit in the use of cover stories is the fact that the organization consciously "plans" to provide disinformation to lower level users--it should not be an ad hoc requirement that necessitates decisions "on the fly" for new cover stories.

Implementation of polyinstantiation adds complexity to database management systems. In addition, there is some concern [7] relating to how users will be able to cope with the complexity of polyinstantiation and understanding the semantics of multiple instances.

Trojan Horse Challenge. This challenge is protecting from malicious code operating without the knowledge of an authorized user. The *-property of the Bell-LaPadula model [2] was designed to prevent unauthorized flow of data by Trojan horses. For example, a program which is reading a Secret file could write Secret data to an unclassified file. The *-property prohibits this occurring by not allowing a program to "write down" in classification level. In this example, the program could only write to Secret files. The key word in this example is "write." Because this challenge is fundamentally an update problem--adding, changing or deleting data in a file. If one controls which code is authorized to write (i.e. update) data, then you can control the problem. For example, a system where no program is authorized to update any data is very secure from Trojan horses (also not much use). In the same way, a Trojan horse that is not authorized to update any files can do no harm to the data--it is reduced to using timing covert channels to disclose classified information.

Covert Storage Channel Challenge. There are two types of covert channels--storage and timing: covert storage channels result when one process causes an object to be written and another process observes the effect, a covert timing channel results when a process produces some effect on system performance that is observable by another process and is measurable with a timing base such as a real-time clock [8]. Covert storage channels require writing to objects, thus it is reduced to an update challenge equivalent to the Trojan horse challenge.

2.2 Underlying Problems

There are several underlying problems with the way the community is approaching the solutions to MLS requirements which in some part have caused or exacerbated these challenges.

Mechanism Madness. The community is caught up in designing mechanisms (in an abstract environment) without a conceptual framework (that works in the real world of application-dependent needs) upon which to hang the mechanisms. Mechanisms are important and at the operating system level they will provide the solutions. However, a mechanism that works in one environment may not be acceptable or effective in another environment.

Control-less Operational Environment. The de facto standard of an operational environment seems to be one where there is little management control "planned" for which programs are allowed to read and write which data. Behind most examples of threats stated in the literature is an **implicit** lack of control on what users and programs are authorized to do. This type of operational environment is not realistic for today's environment, and it certainly should not be acceptable. Although it is a "worst case scenario" the developers and operators of information system can and must demand a more controlled environment.

Technology Obsessed. We are caught up in using technology to solve all problems and, therefore, have lost sight of using other techniques to help solve security problems. Technology and highly trusted systems are important and have their place. However, technology must depend on external factors to successfully implement secure systems. Specifically one needs to use a combination of other techniques from the personnel and procedural aspects of security as part of the overall system design.

3 A Proposed Solution

The key to providing solutions to these challenges is to provide a operational environment with the appropriate controls and a good system design. The word "system" is used in its broadest sense--all the personnel, automated facilities, manual policies and procedures needed to perform a particular function for the organization. The proposed solution combines existing management and design concepts (Section 3.1) along with a new conceptual framework--both for data access control (Section 3.2) and for an operational environment (Section 3.3).

3.1 Management and Design Concepts

The management and design concepts briefly described in this section will come as no surprise to most readers. But it is surprising that many of the technical solutions proposed assume their absence. The important assertion here is that these concepts should not be considered optional--but they must be a required part (actually the foundation) of the total system solution for multilevel database security.

Data Management Resource. For many years the information systems community (especially the database community) has considered "data" and "information" as an organization resource--a resource which can be costly to acquire and maintain. This means that data and information must be managed as carefully as other resources (e.g., personnel, money). Many large organizations have recognized this need and have established organizational elements, such as a database administrator, with specific responsibilities for data management. The significance of this notion for security is the following: they establish which organizational elements (or even specific personnel) are responsible for updating (creating, modifying, and deleting) each data element in the "corporate database." The concept that any user or program can come along and update the data is not acceptable--changing data is controlled. When security becomes a major consideration a database security officer needs to assist in establishing the policies and procedures for creating, updating and querying data in the corporate database.

Design Considerations. As Teresa Lunt states [14] many database security problems can be dealt with effectively, not by requiring new technology in a DBMS, but by smart design of the database. Design considerations will also affect security when taking a system-wide perspective (including the manual procedures, personnel, and other non-technical aspects of the system).

System-wide Perspective. The system that is being designed and engineered must include not just the hardware and software, but also the users and procedures. The database security challenges discussed above go away (or at least become manageable) if the scope of the solution is expanded to include personnel and procedural considerations. For example, in many situations, the need for polyinstantiation can be negated through the use of these considerations. (Section 4 contains specific details.)

3.2 Conceptual Framework for Data Access Control

Like the basic concepts, the reader will look at this section and say, "so what's new?" The answer is not that this is a revolutionary concept, but that in a database environment this framework should be the de facto standard and not optional to provide a satisfactory foundation for a secure system. The conceptual framework for access control involves a series of increasingly stringent criteria as shown in Figure 1. The top two levels involve disclosure access controls. The first level represents trust as implemented in the mandatory access control (MAC) hierarchical levels. The next level, the size of which reflects a smaller number of users, represents need-to-know requirements. Note that both non-heirarchical MAC categories and organizationally-managed discretionary access control (DAC) mechanisms can be used to enforce this level of access control.

Fortunately, the concepts of need-to-know and hierarchical levels of trust are well established and reasonably well understood. Unfortunately, that is not the case for access control for the integrity issues of update, creation, and deletion. The last two levels of the framework involve access control address integrity issues. As shown graphically, a much smaller population of users should be authorized to update data based on *need-to-change* requirements. In a similar manner, an even smaller number of users should be authorized to create or delete data (either to add/delete instances of entities, e.g. adding a new employee, or to add/delete files or relations to the database). The need-to-create/delete level could be either a subset or disjoint set of the need-to-change level.

Unfortunately, the concepts of need-to-change and need-to-create are not well established even though Biba [3] introduced the equivalent notion of *need-to-modify*. More recently Jueneman [12] used the term *need-to-do* which includes both the update and create functions.

To summarize, many users may be trusted to access certain data, while only a subset of those users will be authorized to access the data based upon need-to-know criteria. An even smaller subset of users should be authorized to update data based upon a need-to-change policy, with only a few users authorized to create or delete data. Once again an important point--this access control framework for a database environment should not be considered just as something that could be implemented; it must be provided as the policy

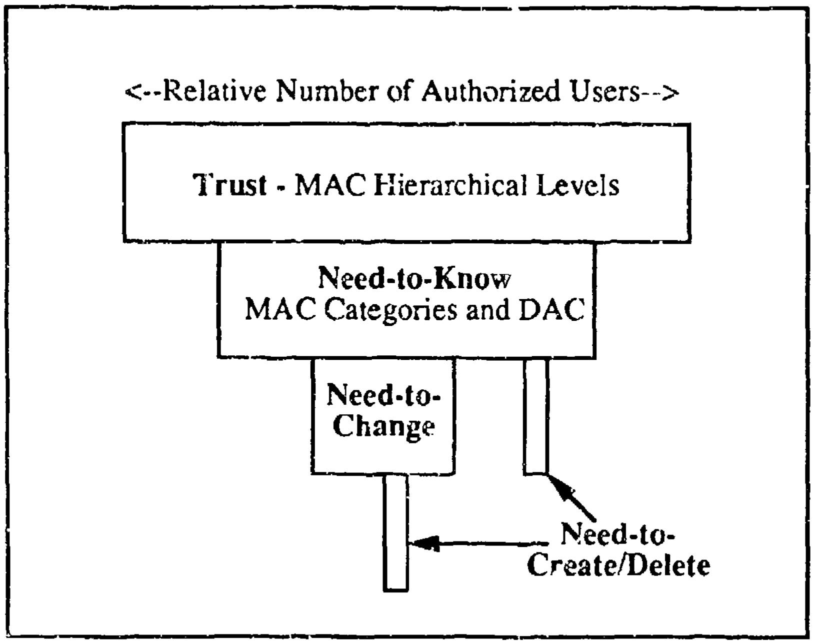

<--Relative Number of Authorized Users-->
Trust - MAC Hierarchical Levels
Need-to-Know
MAC Categories and DAC
Need-to-
Change
Need-to-
Create/Delete

Figure 1: Access Control Conceptual Framework

that will govern the design and implementation of the information systems that use corporate data. Need-to-change, need-to-create, and need-to-delete policies must be explicitly stated for each and every data element in the corporate database.

3.3 Conceptual Operational Framework

Unfortunately, the entire world is not all wonderful--there is malicious code; there are real threats to information systems. Figure 2 provides a conceptual operational framework that incorporates the concepts presented above with the realities of a less-friendly environment. There should be two fundamental domains of the operational system--the part under control of the organization (database administrator, security officer, and software developers) and then the rest of the system over which there is a much lower a prior level of control. The operating system (OS) trusted computing base (TCB) boundary represents facilities an MLS OS provides to mediate access between subjects (i.e., programs) and OS objects. Programs which are part of the OS TCB are not shown.

The concepts described above (Section 3.1 and 3.2) are mandatory for the controlled environment, i.e., inside what we call the *control boundary.* The environment outside the control boundary is a less-privileged domain. It conforms to an environment where there is little control over which users invoke which programs that may contain unknown amounts of malicious code. (This is the only type of environment normally assumed to be present.) This domain is limited to read-only programs for data objects within the control boundary, but may contain programs that update data objects that are outside the control boundary.

Inside the control boundary is a privileged-domain which has two types of objects: data objects (files relations, etc.) that contain corporate data; and data management-relevant programs that are authorized to update the data objects within the control boundary. If one was to add users (authorized to execute the data management-relevant programs within the control boundary) to Figure 2, there is an obvious parallel between the triples of the Clark-

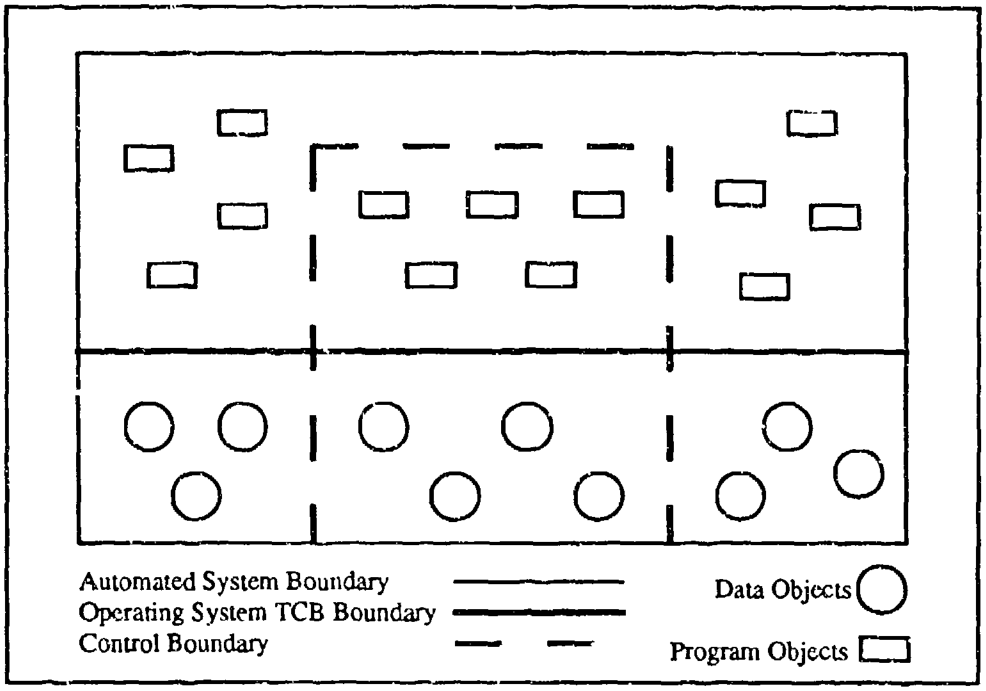


Figure 2: Conceptual Operational Framework

Wilson model (constrained data item, well-formed transaction, and user-id) [4] and the objects within the control boundary.

Since most of the difficult challenges of Section 2 are update problems, if one strictly controls the update function (i.e., the programs that are authorized to update) the problem is effectively solved. To do that, only specified data management-relevant programs within the control boundary are authorized to update corporate data objects that are within the boundary. Conceivably the data management-relevant programs objects would include both an MLS DBMS (trusted to an appropriate level) and programs which are trusted to properly update data elements (i.e., have no Trojan Horses). Until such time as that DBMS can be built where the entire update mechanism can be trusted to be free of Trojan horses, the update programs will need to be code separate from the DBMS. The MLS DBMS then becomes only a retrieval system--but a very powerful one that mediates read access to data (both mandatory and discretionary controls) at a database level of granularity (data elements, tuples, records, objects, attributes, etc.) as opposed to an operating system level of granularity (e.g. files).

The verification that application programs are free of malicious code is difficult and a subject of current research. Much work remains to be accomplished; however, keeping the programs small by isolating code that updates specific data elements (or groups of elements) would seem to be workable as a partial solution. The requirements for certification and/or accreditation of this approach will also be difficultn.

Having only "good" programs update the corporate data is necessary, but not sufficient. One must also keep the "bad" programs from getting at the corporate data as well as keep unauthorized users from executing the good programs. The underlying MLS OS must enforce this separation through mandatory access controls. The MLS OS prohibits all programs (malicious or benign) outside the control boundary from updating data objects within the control boundary. The OS also prohibits unauthorized users from executing programs within the control boundary. The OS performs another important function--it uses mandatory access controls to ensure all read accesses to data objects within the control boundary go through the MLS DBMS (or other authorized programs within the control boundary).

The advantage of this approach described above is that the rules for within the control boundary can be different than those for outside. Specifically, since the environment within the control boundary is fairly benign, there is a greater possibility that classifying and downgrading of data as it goes out of the system boundary can be automated [15].

As a side note, the control boundary and system boundary are presented to represent a single, monolithic computer system. However, this concept also can be applied to distributed systems where the control boundary spans several computer systems. Although possible, the technological challenges for assuring control across multiple systems are significant.

4 Applying the Solution

The following paragraphs illustrate how the challenges of Section 2 can effectively be managed using the management and design concepts, conceptual framework for access control and conceptual operational framework of Section 3.

Polyinstantiation (the update challenge). When considering keeping the user from gaining information, this challenge is solved with the simple application of procedural security and good system design. Trusted update programs are even not required. Specifically, only selected users are authorized to update specific data element--and those users have the correct clearance to update those data elements. This means that if instances of a data element can be classified at multiple levels (e.g., Unclassified through Top-Secret) then the small number of users authorized to update that data element must have a Top-Secret clearance. Since all users authorized to update the data element are authorized to see all the data, there is no reason to ever need to polyinstantiate. The level of trust necessary to qualify the update program for inclusion in the control boundary is required to ensure that a Trojan horse cannot obtain unauthorized information.

Polyinstantiation (the cover story challenge). The cover story challenge can be solved using database design techniques. Two data elements are--one for the real value and one for the cover story. In the flight example, Secret and lower level users would see the data element "destination" which is really a cover story while the Top-Secret users would see the data element "real-destination." As an option the Top-Secret user might need to see both data elements. For this approach to be effective, cover stories must be a conscious part of the design of the system (as opposed as being made up at execution time). We believe this is a reasonable constraint.

Integrity. Using the conceptual framework of Section 3.2, only a small number of users (with the correct clearance) would be explicitly authorized to update (create, modify or delete) specific data elements--and users are authorized for **each** data element in the corporate database. In essence, the organization controls (through the database administrator/security officer) exactly which user(s) can update each data element with specified programs.

Trojan Horses. Since this challenge is an update problem, if one controls which programs can update corporate data elements, and ensure those programs have no Trojan horses, then you eliminate the challenge. We recognize that the methods to provide (certify, accredit) Trojan horse-free programs are not simple to implement. The task is made smaller, when only concerned with code that updates, rather than all code.

Covert Channels. Once again, storage covert channels are an update challenge with the same solutions as for Trojan Horses--if it cannot write anything, then it cannot pass information. Timing covert problems are another matter--the concepts described above cannot solve this challenge.

5 Conclusions

The community's orientation towards technology and mechanisms seems to be to the exclusion of more basic approaches to security such as management controls, good system design, and procedural security. Many of the multilevel database challenges are really "update" problems. These challenges facing the database security community can in many instances be solved (and in the other instances made manageable) by incorporating non-technical aspects of security along those technical mechanisms that either are now, or will soon be, available. A conceptual framework, such as that proposed in Section 3, should be the starting point for information systems design--the organization must mandate the proper management controls and procedural security, a need-to-change policy must be stated to ensure data integrity, and an operational environment must be established which supports a more-privileged domain for update programs.

References

[1] Air Force Studies Board Committee on Multilevel Data Management Security, *Multilevel Data Management Security*, National Academy Press, 1983.

[2] Bell, D. E. and LaPadula, L. J., *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MITRE Technical Report, March, 1976.

[3] Biba, K. J., *Integrity Considerations for Secure Computer Systems*, MITRE Corporation Technical Report, April, 1977.

[4] Clark, D. D. and Wilson, D. R., A Comparison of Commercial and Military Computer Security Policies, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April, 1987, pp. 184-194.

[5] Davison, J. W., Implementation Design for a Kernelized Trusted DBMS, *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December, 1988, pp. 91-98.

[6] Denning, D. E., T. F. Lunt, Schell, R. R., Heckman, M., and Shockley, W., A Multilevel Data Model, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April, 1987, pp. 220-234.

[7] Denning, D. E., Database System Lessons Learned from Modeling a a Secure Multilevel Relational Database System, *Database Security: Status and Prospects*, Landwehr, C. E., ed., 1988, pp. 35-44.

[8] Gasser, M. *Building a Secure Computer System*, Von Nostrand Reinhold Company, 1988.

[9] Gray, J. W. and O'Connor, J. P. Jr., A Distributed Architecture for Multilevel Database Security, *Proceedings of the 11th National Computer Security Conference*, December, 1988, pp. 179-187.

[10] Haigh, J. T., Stachour, P. D., Dwyer, P. A., Onvegbe, E., and Thuraisingham, B. M., *Secure Distributed Data Views-Implementation Specification for a Database Management System*, Honeywell Technical Report, May, 1989.

[11] Hinke, T. H. and Schaefer, M., *Secure Data Management System*, System Development Corporation Technical Report, June, 1975.

[12] Jueneman, R. R., Integrity Controls for Military and Commercial Applications, *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December, 1988, pp. 298-322.

[13] Knode, R. B, and Hunt, R. A., Making Databases Secure with TRUDATA Technology, *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December, 1989, pp. 82-90.

[14] Lunt, T. F., Aggregation and Inference: Facts and Fallacies, *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, May, 1989.

[15] Smith, G. W., Classifying and Downgrading: Is a Human Needed in the Loop, *Research Directions in Database Security*, Lunt, T. F. ed., forthcoming.

[16] Wilson, J., A Security Policy for an A1 DBMS (a Trusted Subject), *Proceedings of the 1989 IEEE Symposium on Privacy and Security*, May, 1989, pp. 116-125.

[17] Wood, T., A B2 Secure Database Machine Kernel for Non-Proprietary Hardware, *Proceedings of the 12th National Computer Security Conference*, October, 1989.

A TRUSTED DATABASE MACHINE KERNEL FOR NONPROPRIETARY HARDWARE

Tim Wood
Sybase, Inc.
6475 Christie Ave.
Emeryville, CA 94608
tim@sybase.com

July 12, 1989

1. Abstract

This paper gives a high-level overview of the system architecture of the Sybase Trusted SQL Server, targeted at the B2 level of trust. The Trusted SQL Server is a physical machine control program that is a hybrid of a secure, high-performance DBMS server with a dedicated kernel of original design. The kernel controls all operations of the host hardware and takes the place of the operating system. By replacing the operating system level, the dedicated kernel reduces complexity of the overall system to facilitate evaluation at the B2 level of trust. The DBMS implements its own security policy; the kernel utilizes hardware protection mechanisms to strengthen the assurance that the security policy is not violated. The current implementation runs on a subset of the DIGITAL VAX computer line.

2. Introduction

Today, the client/server model of computing, which connects suppliers of computing resources (server machines) with consumers of those resources (users on client machines) via networks, is being adopted widely in the general user community [DR]. The economic advantages it offers in terms of reduced replication of resources, centralized management of and decentralized access to resources have become evident. Until recently, there have been few trusted DBMS announced, and none based on the client/server model. The Sybase Trusted SQL Server converts a VAX computer into a high-performance database engine, designed to meet the requirements for secure handling of multi-level data in a client/server computing environment.

The Trusted SQL Server has three major design goals:

Security.

The system enforces reference-monitor-based labeled mandatory security on subjects, e.g. user processes and internal subjects such as audit, and objects, e.g. records. Since storage objects other than records are themselves represented by records in the data dictionary, mandatory access control extends to any entity that can be associated with a record in the data dictionary.

Compactness.

Since the Trusted SQL Server is not supported by an operating system, but instead replaces the operating system functionality, it implements required functions, and only those functions, in a relatively small aggregate amount of code. This attribute, coupled with a modular, least-privilege architecture, facilitates the task of assuring the mandatory TCB properties of tamperproofness, nonbypassability and unconditional invocation [TCSEC]. It also offers a path to evaluations beyond the B2 level.

Performance.

The Trusted SQL Server had its genesis in the Sybase SQL Server currently sold in the general commercial market on a range of computing platforms. That system is designed for high transaction throughput and high availability, and to be scalable across computing platforms of varying price/performance ratios. Those features are preserved in the Trusted SQL Server architecture. Absence of any operating system in the Trusted SQL Server machine environment offers complete engineering control over how operations at all levels of abstraction are implemented, hence the opportunity to optimize the operations for DBMS needs. The Trusted SQL Server is designed be tailorable to several specific configurations across the VAX price/performance spectrum.

This paper surveys various architectural features of the Trusted SQL Server that contribute to the realization of these goals. It expands on the work published in [ROUGEAU]. Architectural support for security policy enforcement is the primary focus.

3. System Architecture

Certain terms are defined here for the purposes of this paper: a "module" is a unit of software logic that performs a function or set of functions and has fixed and definite input and output interfaces. A "domain" is a hardware privilege state. A "segment" is a contigu is region of virtual pages that have the same hardware protection codes. A "process" is a set of execution state information with respect to a program. An address space is a set of segments that are addressable in some domain by the current process. An "object" is an atomic unit of data, either a scalar or a data structure, that is directly accessible by exactly one module. In this paper, use of the word "object" intends the definition just given. This definition is distinct from "security object". Not all objects are security objects. Only objects exported by the reference monitor are security objects, and these are labeled by definition. Where security objects are discussed, they are named as such.

The system is composed of two types of code, untrusted code and TCB (trusted computing base) code. The untrusted code is a single module. The TCB code is composed of two major modules, the Policy module and the Kernel module. This paper will discuss selected functions in the TCB interface and internal to the TCB and the mapping between Trusted SQL Server modules and privilege domains.

4. Untrusted Module

It was decided to place the SQL parser and query-plan compiler in the untrusted module. The alt native of placing them within the TCB would require allocating high privilege and trust to these modules. Since the Trusted SQL Server is designed from a uniform assurance approach, a majority of the lower-level modules, or the remainder of the DBMS as in [KEEFE], would require trust. This would violate the design goal of compactness.

The approach in [KEEFE] uses query modification based on schema-specific rules to enforce a security policy. That approach would preclude the re-use of previously-compiled query plans, because the compiled plan form would be based on the security level of the subject on whose behalf it was compiled. Hence, a high user executing a plan compiled by a low user would see results restricted based on the low level. A low user should never be able to execute a plan created by a high user, since that would reveal its existence and form a covert channel.

By placing the parser and compiler outside of the TCB, we have simplified the task of assuring the correctness of the TCB and we have reduced its size.

5. TCB Structure

The Trusted SQL Server design places the TCB boundary at the "execute-query-plan" level. The TCB guarantees that the untrusted module cannot cause a query plan that would violate the security policy to be executed. So, the security of the system does not depend on correct functioning of the parser and query compiler. Yet, previously-compiled query plans can be re-used after sanitization. Query compilation is an unprivileged operation which is isolated to run in an unprivileged domain.

The Kernel module performs physical device and memory management, process isolation, and message passing among trusted processes. The Policy module performs all remaining security-relevant functions. There is no interprocess message passing interface available to the untrusted module. The only operations the untrusted module may perform on objects c trolled by the TCB are those implemented in the TCB Interface.

6. TCB Interface

The TCB Interface is defined as a set of abstract operations implemented as system calls. These calls are used by the untrusted module to request services from the Policy module. No interface to the Kernel module is available to the untrusted module.

The untrusted module may not directly access any object managed by the TCB. The untrusted module requests access to TCB objects via system calls. The TCB guarantees that all system call return information is dominated by the subject security level. The TCB prohibits the untrusted module from exporting that information or changing its label. In particular, the TCB communicates all query results to the client and determines result labels.

7. Mapping Between Trusted SQL Server Modules and Machine Privilege Domains

The protection of TCB objects and operations from untrusted module access by means other than the TCB interface is enforced by mapping each object and operation (text subroutine) of a module to a privilege domain implemented by the target machine. Generically, the Trusted SQL Server uses three abstract domains: Unprivileged, Policy and Kernel; the "I/O domain" described in [ROUGEAU] has been renamed to Kernel domain and the "User domain" renamed to Unprivileged domain. The VAX privilege modes in privilege order are User, Supervisor, Executive and Kernel. The abstract Unprivileged domain corresponds to VAX User mode, the abstract Policy domain corresponds to VAX Executive mode, and the abstract Kernel domain corresponds by name to the highest VAX mode. The Trusted SQL Server does not currently use VAX Supervisor mode.

VAX memory access modes are Read and Write, with Write mode implying Read mode. Each page of an address space is marked with the minimum privilege at which it can be read or written, and the write privilege (if any) must be equal to or higher than the read privilege [VAX].

This scheme is used in the architecture to specify five mappings:

(1) All of the executable text of the Trusted SQL Server in all modules is read-only, regardless of execution domain.

(2) Compiler state information, untrusted text and read-only system-wide lookup tables are accessible from the Unprivileged domain.

(3) Most Policy module text and all its objects are accessible only from Policy domain and higher, however the system is structured so that the Kernel module does not directly access Policy objects or text.

(4) The remaining Policy module text is mapped to be directly executable in the Unprivileged domain. This mapping is designated for routines which need to be used by untrusted and trusted modules. These routines must follow a protocol identical to the one that the untrusted module uses to obtain services from more-privileged code. That is, even though these routines are part of the TCB, they have no greater privilege than the untrusted code.

(5) All Kernel module objects and text are accessible only from Kernel domain.

8. System Calls

System calls in the Trusted SQL Server are implemented in familiar ways [KENAH]. A special instruction causes a trap to a trusted handler which manages the details of creating a protected scope for the system routine execution and communicating results (but no other TCB state information) back to the (less-privileged) caller.

9. Processes and Addressing

Up to this point we have discussed objects and text by module. A Trusted SQL Server process associated with a client connection executes untrusted text, Policy text and Kernel text, transferring control between domains via system call and return. It is now meaningful to discuss the structural view a Trusted SQL Server process has of objects and text.

For this discussion, we recursively define "task" as an execution thread or context which shares the address space provided by an operating system with zero or more tasks. In contrast, "processes" are execution contexts with distinct, virtual address spaces defined by the kernel and demarcated in hardware. The commercial SQL Server performs internal multi-tasking and runs as a single process on top of an operating system. The Trusted SQL Server implements true processes.

Any Trusted SQL Server process may only access memory within its address space. All objects and text are located in some address space. The Trusted SQL Server has a simple set of rules that determine what segments in what address spaces can be accessed in what privilege modes.

All process-private data is located in process data space. Segments in the process data space include a process-status structure, a stack for each domain, and query-plan segment. Segments other than the query plan segment are mapped to distinct physical memory pages at process creation time.

10. Object Caching and Sharing

The Trusted SQL Server architecture reconciles two sometimes opposing goals: to provide high-assurance enforcement of the mandatory security policy and to maximize object caching and sharing to achieve high performance.

Certainly, no object sharing scheme can be used if it violates the security policy. The original SQL Server has many object sharing schemes which have been instrumental in achieving high performance. For the Trusted SQL Server, descriptors for recently-opened tables are saved in TCB memory. Mandatory access control is always enforced for the "open table" operation before the search for a table descriptor is made. Since there is no way for the untrusted module to determine whether a descriptor is present in main memory, this sharing scheme does not introduce a covert channel or a security compromise.

The Trusted SQL Server has retained most sharing with no compromise of the security policy. The original sharing schemes that did not conform to the security policy have been modified to conform in the Trusted SQL Server with minimum impact on performance.

Most of the object sharing and caching takes place within the Policy module running in the Policy domain. None occurs within the Unprivileged domain. Since the TCB never exports its objects to the Unprivileged domain, the sharing and caching schemes from the original SQL Server were imported into the Trusted SQL Server with few changes.

The mandatory reference monitor logic sits above all of the sharing algorithms, so no process can make use of a labeled shared object unless the process security label dominates the label of the object.

11. Query Plan Validation

SQL is a non-procedural language. It specifies conditions for retrieving, creating, modifying and deleting subsets and aggregates of the data in the database. The query plan is the procedural specification for performing the operation specified in the non-procedural query.

Query plans are the basic structure by which the TCB receives requests to do work. Query plans are constructed by untrusted code. They can be executed only after first being validated by the TCB. Successful validation transforms a query plan from an untrusted module object to a Policy module object. The plan then may be executed by subjects whose security levels dominate that of the subject process which constructed the query plan. Unsuccessful validation causes the TCB to deallocate the query plan and return with an error indication. This section discusses the meaning of query plan validation and its design motivation.

All query plans are created by untrusted subjects. Query plans may exist for one-time use or repeated use. In the case of one-time use, the query plan is private to its creator and never accessible by other users. In the case of repeated use, the query plan is saved in the database along with the label of its creator.

When saved query text is read in from the database prior to execution, it must be compiled by the untrusted module. The untrusted module receives a copy of the uncompiled plan, creates a compiled plan in the process query plan segment, and submits the compiled plan to the TCB for validation and execution.

The validation step is security-critical. The TCB is receiving a complex data structure, the query plan that specifies database operations, from the untrusted module. It must apply several consistency checks to the query plan to verify that executing it will not subvert the security policy. These consistency checks are done in the validator routines in the Policy domain.

This validation procedure does NOT verify that the query plan is the correct procedural expression of the non-procedural SQL submitted by the subject. Such verification would involve virtual reimplementation of the untrusted compiler logic within the TCB. The validator only guarantees that use and reuse of the query plan will not violate the security policy and integrity of the database.

If the validator finds no errors in the query plan, it labels the plan with the security label of the subject which submitted it and enters the plan into the system "procedure cache". Once this is done, any subject whose label dominates that of the plan may execute it. The physical memory of the machine is used to cache valid, recently-used query plans, so that they need not be read from the database at each execution.

Query plans and the table descriptors mentioned earlier are examples of TCB internal objects. Once they are created, they are never exported to the Unprivileged domain or to the DBMS client. They are labeled in order to use the mandatory access policy to aid structuring of the TCB.

The untrusted module has no way of removing a plan from the cache or determining whether a plan is present in the cache. Thus the procedure cache cannot be used as a covert channel.

Since plans within the procedure cache have been validated, most of validation can be bypassed when the plans are next executed, because validated plans are never made accessible to the Unprivileged domain. It is necessary and sufficient just to sanitize plans between executions to eliminate any leftover state from previous executions. Validation must complete without error in all cases before the plan may be executed. Any failure results in deallocation of the plan and an error return from the TCB.

12. External Interfaces

The Trusted SQL Server communicates with users via two facilities, the trusted interfaces and the master network. The trusted interfaces allow users, system administrators and system security officers to perform security-sensitive operations. The master network is the medium by which users submit database queries and commands from their client nodes and receive results and status.

12.1. Trusted Interfaces

The trusted interfaces are direct user interfaces to the TCB; all operations performed through the trusted interfaces are implemented exclusively within the TCB and execute entirely within the Policy domain. No untrusted module code is ever executed in a trusted interface operation.

A trusted interface is physically implemented as a serial character terminal directly connected to a port on the target VAX. There are usually four trusted interface (TI) devices in an installation.

The first device is the system security officer (SSO) TI terminal. This is used to create, remove or lock user authorizations, drop tables, perform trusted writedown, etc.

The second device is the user TI. This is for use by ordinary users to change their own passwords, define discretionary access to their objects, etc.

The third device is the tape dump console. It is simply an interactive means to synchronize mounting and unmounting of tape volumes with the system dump and load facilities.

The fourth device is the audit printer, which outputs the audit event records in hardcopy form. It is c nfigured in the same way as the other TI devices, but is logically an output-only device. There is no notion of user input to the audit printer device.

All TI sessions are initiated by a login sequence with non-echoed passwords with the exception of the audit printer since there is no notion of a user-initiated session with this device. TI sessions use a simple menu/prompt structure. There is no language interface.

12.2. Master Network

The master network is the medium by which users establish sessions with the Trusted SQL Server, submit queries and receive results. The Trusted SQL Server design allows for more than one "master" network to carry user sessions. In that case, there is no hierarchy among these networks, and the design does not require them to run the same protocol or have the same architecture.

To initiate a connection from a client machine, the user's application sends a message requesting session establishment to a well-known master network socket owned by the Trusted SQL Server. The Server trusted process listening for such requests receives the request and creates a trusted process that responds to the application with a request for a login name, security level and password. The application responds with this information. The Server trusted process checks the login name and password for validity and checks that the given security level does not exceed the maximum level designated fo that user. If any of these checks fail, a uniform "Login Incorrect" error message is returned to the client, regardless of the actual error and the connection is closed. The login failure is audited.

If all checks succeed, the successful login is audited and the trusted process transfers control from the login module running in the Policy domain to a fixed entry point within the untrusted module running in the Unprivileged domain in a secure atomic operation.

13. Covert Channels

Covert channel analysis work is proceeding concurrently with development. We have taken the shared resource matrix [KEMM] approach and are applying it at a detailed design level. Some open issues remain in the area of table and page locking mechanisms, and research into strategies for addressing them is continuing. Finding a strategy that allows system performance goals to be met is an important challenge.

14. Conclusion

The Sybase Trusted SQL Server is the first DBMS to be developed that provides high-performance, multilevel relational database management with high security assurance. Its client/server architecture incorporates mandatory security control with high assurance, compactness and embedded operating system functionality to facilitate evaluation, and scalable high performance to handle the demands of secure transaction processing applications efficiently.

15. References

[DR] J. Fegreus, "Architecture for the '90s," *Digital Review* editorial. p. 32, March 6, 1989.

[KEEFE] T. F. Keefe, M. B. Thuraisingham, W. T. Tsai, "Secure Query-Processing Strategies," *IEEE Computer*, Vol. 22 No. 3, pp. 63-70, March 1989.

[KEMM] Kemmerer, R. A. "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels." *ACM Transactions* *on Computing Systems*, Vol. 1 No. 3, pp. 256-77, 1983.

[ROUGEAU] Patricia A. Rougeau and Edward D. Sturms, "The Sybase Secure Dataserver: A Solution to the Multilevel Secure DBMS Problem," in *Proceedings of the 10th NCSC Conference*, 1987, pp. 211-215.

[TCSEC] Department of Defense Trusted Computer Systems Evaluation Criteria, CSC-STD-001-83, Library No. S225,711, DoD Computer Security Center, Fort George G. Meade, Md, December 1985.

[VAX] *VAX Architecture Handbook*. Order no. EB-19580-20/31, Digital Equipment Corp., Maynard, MA, 1981, p. 111.

[KENAH] Lawrence J. Kenah, Ruth E. Goldberg, Simon F. Bate, *VAX/VMS Internals and Data Structures*. Order no. EY-8264E-DP, Digital Press, Bedford, MA, 1988, ch. 9, pp. 199-204.

THE SEAVIEW VERIFICATION EFFORT

R. Alan Whitehurst
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Teresa F. Lunt
Computer Science Laboratory
SRI International
Menlo Park, CA 94025

Abstract

This paper discusses the verification of the SeaView formal top-level specifications (FTLS) and the benefits that were gained from formally specifying and verifying selected database operations.[1] The SeaView specifications describe a multilevel secure relational database system and were written in the formal specification language of the SRI specification and verification system EHDM. The process of specification and verification substantially improved the quality and completeness of the SeaView design.

1 Introduction

The SeaView project was a three-year program to create the design of a multilevel secure relational database system that met the criteria for Class A1. The project produced a security policy and interpretation [6], a multilevel relational data model that extends the standard relational model to support explicit labels for elements and tuples [4], a formal security policy model [7], formal top-level specifications [9, 10], and implementation specifications for new system components. In addition, a demonstration system that illustrates polyinstantiation and operations on multilevel relations was created.

The SeaView specifications contain a formal policy model of the security requirements for multilevel secure databases and an abstract description of the database operations. As part of the SeaView effort, we attempted a formal verification of the database operations against the security properties of the policy model. In doing so, we discovered that our intuition about the formulation of some of these

[1] The work described in this paper was supported by the U.S. Air Force, Rome Air Development Center (RADC) under Contract F30602-85-C-0243.

operations was often incomplete, and, sometimes incorrect; the final formulation of these operations varied drastically from the original as a direct result of the insights gained in the process of attempting to prove that the operations satisfied the security properties of the model.

This paper describes the verification effort. In Section 2, a general introduction to formal verification is presented which includes the philosophy we adopted during the Seaview verification effort. An overview of the strategy used to approach the verification effort is given in Section 3 and the results of the verification efforts and our observations about the value of this exercise are contained in Section 4.

2 Verification

A reasonable place to begin a discussion about a project which applied formal methods is to ask the question, "Why use formal verification?" To answer this question, it is necessary to view the motivation to use formal methods from three distinct vantage points: policy, economy, and quality.

As stated earlier, the goal of the Seaview project was to design a secure database that would meet the criteria for Class A1. According to the "Orange Book" [1]:

> *"A formal model of the security policy must be clearly identified and documented, including a mathematical proof that the model is consistent with it axioms and is sufficient to support the security policy..."*
>
> *"The FTLS of the TCB must be shown to be consistent with the model by formal techniques where possible..."*

What this means is that to meet the criteria for Class A1, we had to show that the FTLS was consistent with the SeaView security model using formal techniques. While this may be sufficient justification to motivate the use of formal methods in some situations, if there is not a greater benefit, we will probably see very few Class A1 systems. The cost of formally verifying a system is high[2]—if the only motivation for applying formal methods was to satisfy Government requirements, it seems the cost would far outweigh the benefits. We found that the benefits derived from the application of formal techniques in the SeaView design went far beyond the mere satisfaction of Government requirements. We believe that verification can be economically cost effective and is absolutely necessary when dealing with critical software systems.

One of the most serious problems facing the computer industry today is the mushrooming costs of creating and maintaining software systems. Over the past decade, the major costs associated with fielding a computer system have shifted from hardware-related expenses to software-related expenses [5], as shown in Figure 1. The reasons for this reversal are not hard to discern: the advent of VLSI technologies has caused hardware costs to plummet, while the creation of software

[2] See Section 4 for a description of the scope of the effort and the costs involved in the Seaview verification.

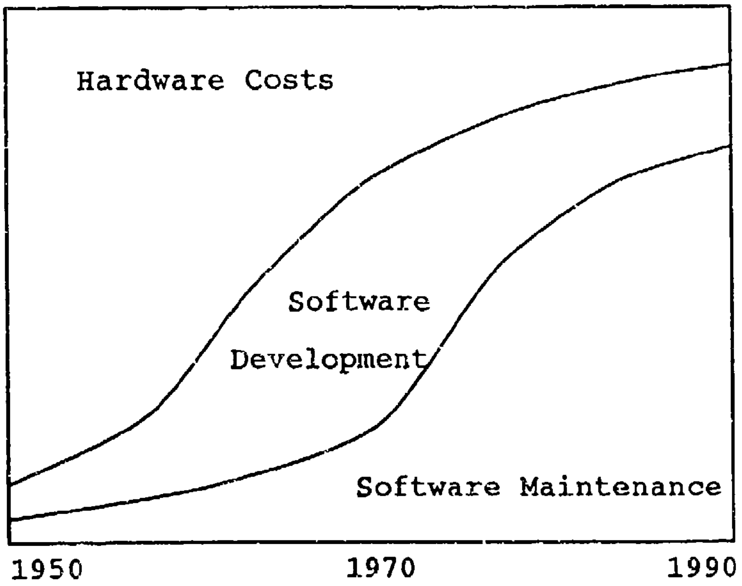


Figure 1: Shifting Costs for Hardware and Software

has remained labor intensive and labor costs have continually increased. However, Figure 1 illustrates that, besides personnel costs, an ever greater share of the expense associated with computer software is consumed by maintenance of existing software systems.

Unlike hardware systems which do eventually wear out, softwere is immune to deterioration. Therefore, much of the maintenance costs spent on software are directed towards the correction of human-induced problems—problems in the design or implementation that either existed when the system was originally created or that were introduced through later maintenance efforts. Furthermore, because of the nature and complexity of software, the cost of error correction increases exponentially throughout each phase of the software lifecycle. According to current figures, roughly 75% of the total cost of fielding and maintaining a computer system could be eliminated if design errors were detected and corrected early in the software development lifecycle—preferably at the point at which they enter the system design.

Irrespective of the economic benefits of being able to produce correct software, there are applications which require nothing less than the highest achievable degree of reliability. One such class of applications are those software systems entrusted with the protection of national security information. Because of the complexity of today's software systems and the deficiencies in existing software development methods, current systems suffer from incomplete or erroneous implementation. Attempts are made to design and implement security features, but often a single mistake can render the entire security controls impotent. We believe that a very tight coupling exists between information security and software quality—the security of a system depends on the correctness of the implementation of its security controls.

Current software development practices rely upon various forms of testing to verify the correctness of the design and implementation. While we believe that testing is absolutely necessary, we also believe that it is insufficient for determining the trustworthiness of systems being used in critical applications. The inadequacy of testing can be summarized by three main observations:

- Testing is capable of finding only the presence of errors, but not in showing their absence.
- Since we cannot show the absence of errors, there exists no criteria whereby we can know if testing is complete. And since it is infeasible to test a system's response to every legal input, testing is always incomplete.

In critical applications like computer security, where a single error (which may go undetected for months or even years) may be able to be exploited to comprise the security of the entire system, it is imperative that methods be sought to augment the assurances provided by testing.

Formal verification is the process of showing mathematically that a correspondence exists between two levels of abstraction in a design hierarchy—in other words, between a specification of relevant properties and an implementation of that specification. Showing that a correspondence exists establishes that the properties expressed in the specification are preserved in the implementation. In Class A1 systems, it is required to show that a correspondence exists between a mathematical statement of the critical properties, often referred to as a security model, and the top-level design specifications, or FTLS. Although it is not required, the process can be repeated using the FTLS as the abstract specifications to establish correspondence between the FTLS and a more detailed level of specification. This can be repeated as many times as necessary until a correspondence is established between some intermediate specification and the actual implementation in hardware and software, as illustrated in Figure 2.

Formal verification is a method for the systematic development of reliable software. It is currently the only method available for establishing the absence of certain errors in a software system. Although the cost of formal verification may seem high, the technology, when mature, may offer the possibility of the creation of provably correct software—software that corresponds unequivocally to its requirements. In formal verification, the effort of creating error-free systems is shifted from the testing and maintenance phases to the design and implementation phase, where that effort is more cost effective. Formal verification may well prove to be more economical than conventional testing alone when the costs of maintenance are viewed over the entire life of the software system.

3 The SeaView Verification

This section outlines the strategy used in verifying the security of the SeaView database operations in the FTLS with respect to the properties expressed in the

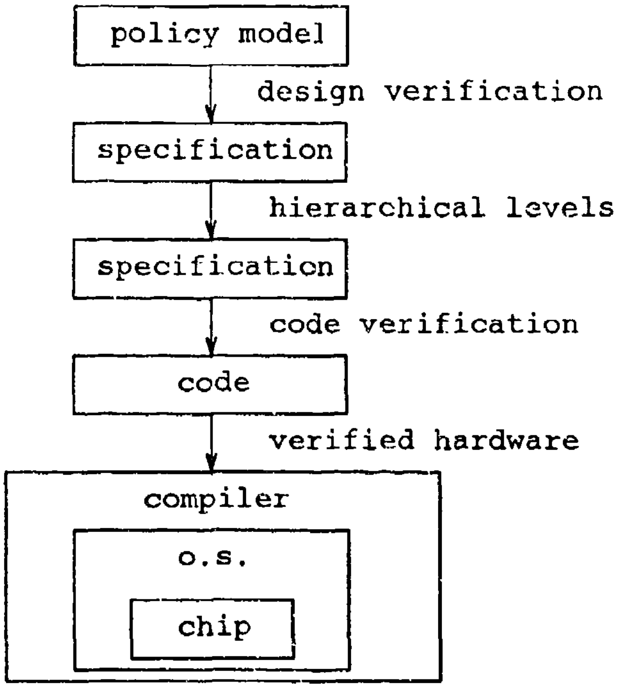


Figure 2: Verification of Developmental Hierarchy

SeaView model.

SeaView has defined a standard implementation-independent multilevel query language called MSQL (Multilevel Structured Query Language) for defining and manipulating multilevel relations [8]. MSQL allows users to retrieve and/or modify data based on their classifications. It includes facilities to allow users to deal with polyinstantiation. It enforces a set of integrity rules on multilevel relations.

The SeaView FTLS specifies the functionality of the SeaView MSQL interface, with definitions of operations for creating and manipulating multilevel relations. There are 31 MSQL operations specified in the FTLS. The functional specification of the MSQL operations was designed to provide a foundation for a later design and implementation effort. For this reason, it is very important that the operations be correctly defined or, more precisely, that the operations as specified satisfy the state and transition properties of the SeaView security model [7]. To satisfy these properties, the execution of an operation must terminate in a secure state if it began in a secure state, and the transformation from state to state must be well behaved. Ideally, this should be shown for all operations; however, verification was not the primary purpose of the Seaview project and resources were constrained. In order to maximize the benefits of the verification effort, we constructed proofs for only a two-element subset of the operations:

- create_mreal_relation
- update_real_tuple.

We chose these particular operations because they are interesting and sufficiently dissimilar that we expected them to trigger scrutiny of different parts of the SeaView specification; between them they stress most of the properties of the SeaView model. Moreover, they are, in our estimation, two of the most difficult to verify. After completing the verification of these two operations, we believe that the quality and completeness of the SeaView specifications were greatly increased. We also believe, however, that any implementation based on the SeaView model would greatly benefit from the completion of the verification for the remaining MSQL operations.

The specifications of the MSQL operations are expressed in the form of pre- and post-conditions. Pre-conditions specify what must be satisfied before a particular operation can be invoked. Post-conditions express the changes to the system state caused by execution of the operation and specify what properties must hold after the operation has been executed in order to satisfy the state and transition security properties of the SeaView model. Details of the MSQL operations may be found in the formal specifications [9].

One of the benefits of conducting formal analysis is that the designers are forced to make their assumptions explicit—assumptions that would otherwise escape scrutiny. In the SeaView specification, we found we had made many implicit assumptions about which objects the operation would *not* affect. Although these assumptions were not stated in the original specification, they were necessary to complete the proofs, and were added to the final specification.

For example, in order to complete the proof of security for create_mreal_relation, it was necessary to augment the post-condition of create_mreal_relation with several additional assertions that reflected the following assumptions: (1) the current access set for the new object would not be changed by the creation function, and (2) the values of all other relations would not change. These assertions were not included in the original specification because they seemed so intuitively obvious, but it is this type of assumption that often causes trouble in later design and implementation phases because it goes unrecorded. While it may seem improbable that anyone would implement an operation to create relations that simultaneously changed the value of other relations, we cannot leave the security of the system up to the "reasonableness" of the designer or implementer. Furthermore, there are situations in which even reasonable decisions may violate the assumptions of the security model. If we had not attempted to prove that the operation satisfied the security properties, we would not have noticed that these assumptions had been omitted from the original specification. As a result, the security of the system could have been compromised if later design and implementation decisions were made contrary to these unstated assumptions.

The update_mreal_tuple operation was more complex than the create_mreal_relation operation. We found that our intuition about the pre- and post-conditions of this operation was also incorrect. For example, the original specification had omitted the very important constraint that primary key attributes could not be updated (so that we could not prove that the entity integrity property[3] was preserved by this operation). The final formulation of update_real_tuple, as a result of the verification process, varies drastically from the original.

Besides changes made to the specification of the MSQL operations, several of the properties of the model, such as the referential integrity property[4], were altered and improved due to the scrutiny of the verification process. There were also several missing properties which were discovered, such as the property that real relations must have no duplicate tuples. These necessary changes were discovered in the process of attempting to prove that the operations satisfied the security and integrity properties of the model.

We believe that the major benefit from using formal methods is derived through the proof effort– an effort in which an automated verification methodology (like SRI's EHDM [2]) plays the part of a "skeptical antagonist" that demands justification for every step in the chain of reasoning. The benefit from undertaking such an endeavor is not in the achievement of a finished proof, but rather in the added insight derived from going through the proof process. Therefore, while proof aids that increase productivity are appreciated, any tool that lessens the responsibility for understanding the proof under construction is self-defeating. For this reason, although EHDM contains a proof checker, a proof-building aid and a fully automated prover, we preferred to use only the proof checking mode of EHDM.

Our usual approach to the construction of a proof was to work each proof out by hand outside of the verification environment, presenting the finished proof to the verification system only for validation. Often our proofs would fail. It was in these instances that we gained the most insight about our application. In analyzing why a proof was unsuccessful, we often detected inaccuracies or incompleteness in our specifications. We believe it is exactly this process which benefited the SeaView design.

We also encountered proofs that we were successful in proving but that, in retrospect, we convinced ourselves should have failed. When these proofs succeeded, it pointed to errors in our specifications of the model properties or the operations, or to an incomplete or incorrect understanding of some aspect of the model. Based on our experiences, we believe that it is insufficient to do a proof without understanding why the proof succeeded or failed. This understanding is pivotal to the realization of any expected benefits from the verification process.

4 Verification Results

The SeaView specifications and proofs are structured into 102 modules, comprising over 9,100 lines of specification. The SeaView model is contained in 35 specifica-

[3] the entity integrity property states that all values used as keys must be non-null

[4] the referential integrity property states that only data that exists in the database may be referenced

tion modules, while the MSQL operations require an additional 7 modules. The remaining 60 modules constitute the proof of security for the two operations that were analyzed. In all, 255 proofs were constructed with 275 man-hours of effort.

The exercise of formally specifying the SeaView properties in the specification language of EHDM [3] resulted in our discovering and clarifying many ambiguities and imprecise or incomplete statements of the properties described in the model report. Through this exercise we also identified and corrected many mistakes in the properties of the model. We should also note here that the Class A1 requirement (that the model's properties be verifiable) was a strong influence in our formulating the model properties so as to be amenable to proof. We struggled particularly with the transition properties, which went through many iterations before we were satisfied with their verifiability.

The exercise of formally specifying the operations on multilevel data in EHDM's specification language resulted in a much more complete understanding of those operations than would otherwise have been the case. Although we thought we had a good idea of the conditions the operations had to include in order to satisfy the properties, we discovered in the process of specifying the operations that our intuition was incomplete, and in every case additional pre- and post-conditions had to be added to the operation specifications. The exercise of formally proving that the operations satisfy the properties has uncovered many additional errors and omissions in the specification of the two operations that were formally verified and in the specification of the properties in the model. Several additional properties were added to the SeaView model and many model properties were modified as a result of the analysis the model underwent in the verification process. Thus feedback was provided that increased the strength and completeness of the model itself—feedback that would have been nearly impossible to gain any other way.

Likewise, the EHDM verification system received feedback during the course of the verification effort which resulted in the identification and repair of several implementation problems. While these problems were minor and did not affect the soundness of the formal analysis, still the robustness of the implementation improved because of the SeaView effort. Furthermore, through the experience of the SeaView verification effort, additional features were proposed for addition to EHDM expressly for supporting large-scale verification efforts.

We believe the use of the EHDM system as the basis for the specification and verification of the SeaView model has contributed greatly to the success of the verification effort. The elegance and expressiveness of the EHDM language simplified the initial effort of formulating the specifications and left us free to consider the semantics of the SeaView properties, rather than the syntax of the specification language. It also allowed us to work at the highly abstract level of the FTLS in a manner that seemed natural and was well supported. The integrated environment, with support for configuration management, and the ability to structure the specifications into modules with explicit interfaces to other modules, greatly reduced the complexity of managing such a large specification. EHDM also provides support tools that help the user in configuration management of large numbers of interrelated specification modules—monitoring when specification modules are changed and keeping track of the effects of those changes on dependent modules.

The specification of the MSQL operations was intended to provide a foundation for later design and implementation efforts based on the SeaView model. The increased understanding of the two verified operations, with the corresponding increase in the quality and completeness of the operational specifications, leads us to conclude that completing the verification of the remainder of the MSQL operations would be of immense benefit to any project using SeaView as the basis of its design. Verification of the remaining properties should not be as expensive as for these first two, because a significant portion of the original effort was directed at working out the paradigm and proof strategies. Moreover, the improvements made to the SeaView model properties as a result of the verification of the initial two properties will make the verification of the remaining properties go more smoothly. Although we formally proved only two of the SeaView database operations, many of the flaws we discovered in the specification of those operations were applicable to entire classes of operations, and we updated the formal specifications corresponding. Therefore, we would expect the work of verifying the remaining operations to go much faster.

5 Conclusions

The benefit we obtained from doing this exercise was enormous [10]. During the process of constructing the proofs, we found many areas in which our initial specification of the SeaView operations was faulty, we discovered errors in the statement of the SeaView security properties, and we discovered "missing" security properties that were needed. Completion of the SeaView verification could lead to additional such discoveries. The SeaView design benefited greatly from the increased scrutiny and analysis of the verification process.

References

[1] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria.* Technical Report DOD 5200.28-STD, Department of Defense, December 1985.

[2] J. S. Crow, S. T. Jefferson, R. Lee, P. M. Melliar-Smith, J. M. Rushby, R. L. Schwartz, R. E. Shostak, and F. W. von Henke. *SRI Specification and Verification System Version 3.1 - User's Guide.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, October 1986.

[3] J. S. Crow, S. T. Jefferson, R. Lee, P. M. Melliar-Smith, J. M. Rushby, R. L. Schwartz, R. E. Shostak, and F. W. von Henke. *SRI Specification and Verification System Version 3.0 - Preliminary Definition of the Revised SPECIAL Specification Language.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, May 1986.

[4] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley. A multilevel relational data model. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April 1987.

[5] Richard E. Fairley. *Software Engineering Concepts. McGraw-Hill Series in Software Engineering and Technology*, McGraw-Hill, Inc., New York, 1985.

[6] T. F. Lunt, D. E. Denning, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. *Final Report Vol. 1: Security Policy and Policy Interpretation for a Class A1 Multilevel Secure Relational Database System.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, 1988.

[7] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. *Final Report Vol. 2: The SeaView Formal Security Policy Model.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

[8] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. Toward a multilevel relational data language. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December 1988.

[9] T. F. Lunt and R. A. Whitehurst. *Final Report Vol. 3A: The SeaView Formal Top Level Specifications.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

[10] R. Alan Whitehurst and T. F. Lunt. *Final Report Vol. 3B: The SeaView Formal Verification: Proofs.* Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

An Interactive Approach to Ada Verification [1]

Carla Marceau and C. Douglas Harper
Odyssey Research Associates
301A Harris B. Dates Drive
Ithaca, NY 14850
(607) 277-2020

June 29, 1989

[1] This work has been sponsored by the USAF, Rome Air Development Center, under contract number F30602-86-C-0071.

Abstract

Penelope is a prototype Ada verification editor whose user interactively and concurrently develops specifications of programs, their Ada text, and proofs of their verification conditions. With each incremental change the user makes to the specifications and program text, Penelope recomputes and displays the verification conditions. Linked to the syntactic constructs of Ada, these verification conditions and their incomplete proofs guide the user, showing where further development is required, or where an error has been made. A completed proof indicates that the portion of the program in question is complete and correct. Within these proofs, the user may appeal to previously formulated axioms and lemmas in proving and simplifying verification conditions, separating their mathematical content from their program-specific content, and concentrating on the latter.

1 Introduction

Program verification systems hold out the promise of enabling users to formally specify what a program is expected to do and to prove that the program meets its specification. In practice, however, verification systems are still far from the point at which they would become attractive to use for real program development. One reason for this is that in most systems it is hard for the user to relate failures in his proof to specific bugs in his program (or specification), or to reverify a corrected program without starting over. The difficulty of proving verification conditions that are generated fo· even rather simple programs magnifies the frustrations of current non-interactive approaches.

1.1 An interactive approach

The Penelope verification editor was developed to explore a more interactive approach to program verification in the context of a complex programming language, Ada. In a typical verification system, the user writes a specification and a program, and inputs them to a verifier. The specification includes input and output conditions for each subprogram. The verifier produces a verification condition: a candidate theorem which if proven guarantees that the program meets its specification.[1] However, if the programmer cannot prove the verification condition, which is typically the case, he faces a difficult problem. Does the fault lie with the program, or with the specification, or with the proof? And how can he find out? We believe that the magnitude of this problem can be considerably reduced by allowing the programmer to inspect the verification condition as it is generated, to inspect intermediate steps in generating the verification condition, and to verify small pieces of his program. The programmer would like to proceed iteratively as follows:

- write a specification;
- write a crucial piece of code;
- inspect the verification condition immediately, looking for errors; and
- modify the code (or specification) to reflect his improved understanding.

This implies that the programmer should be able to inspect a verification condition corresponding to an incomplete program. It also suggests that a number of small verification conditions closely associated with program constructs would be most useful in isolating errors in the program. The Penelope verification editor is designed to permit just this kind of interactive use.

The programmer uses Penelope to create programs in a subset of Ada, currently including loops, go to statements, user-defined exceptions, subprograms (including recursive subprograms), and packages, but excluding tasking, private types and generics. The editor performs static semantic checking, including resolution of operator overloading. Verification conditions are generated according to the model of Gries [11] and Dijkstra [5]. That is, verification conditions are generated for each subprogram and loop, as well as for certain other syntactic constructs. For subprograms, the user specifies input and output conditions. Statements of the program (in Ada this includes exception handlers and declarations) are viewed as transforming the output condition into a precondition that must be shown to hold in the

[1] Penelope is currently able to provide verification conditions for *partial* correctness, that is, the program meets its specification if it terminates at all.

state in which the program is invoked. The verification condition for a subprogram basically states that the input condition of the program is sufficient to prove this precondition of execution. The editor generates verification conditions immediately, in the same manner as a spreadsheet program, which recomputes value; as the user enters data, so that they may be inspected as the program is developed. The user can prove the verification conditions using a proof editor that is built into Penelope.

1.2 Implementation and use

Penelope is implemented using the Synthesizer Generator [19], which is a generator of syntax-directed editors based on attribute grammars. Penelope has been used to verify Bell LaPadula security properties of a part of the ASOS operating system [21]. It has been used to prove such programs as binary search, greatest common denominator, the *tak* [20] function and other "toy" but non-trivial programs.

1.3 Organization of the paper

This paper will discuss the various aspects of using Penelope: specifying a program, developing a program interactively, and proving verification conditions generated by the editor. We will look in some detail at an example of developing a verified Ada program using Penelope.

2 Specifying a Program

In Penelope specifications are based on the Larch approach to specification [13]. In this approach, developed at MIT, the Larch Shared Language is used to develop mathematical theories (for example, integers, lists, and so on are axiomatized in the Shared Language), while Larch Interface Languages associate specific programming constructs in particular languages with the underlying mathematics. Larch/Ada [1] is a Larch Interface Language developed at Odyssey Research Associates; Larch Interface Languages also exist for Pascal and other languages. The syntax of Larch/Ada follows the syntax of Anna [7], but its semantics is based on a denotational semantics for a close approximation to sequential Ada[2]: Larch/Ada is used to make statements in first-order logic about the denotation of programs. (For a complete description of the mathematical foundation for Penelope see Ramsey [17] and Polak [15].)

Specifying a program in Penelope consists of writing Larch/Ada *annotations* for the subprograms. A specification is a statement of what conditions must hold on entry to and exit from the subprograms. Other kinds of annotations are a convenience to the programmer: these include loop invariants and assertions embedded in the program.

There are several kinds of subprogram annotations that together form the subprogram specification: IN and OUT annotations, RETURN annotations and *propagation* annotations. An IN annotation states an assumption that a certain predicate holds at entry to the subprogram. An OUT annotation states that a certain predicate must hold when the subprogram terminates normally. For example, assume that there is a table privilege_table from whose state one can see whether a property called system_is_secure holds. Then the following specification asserts that on entry to the subprogram we can assume that the table is in a state such that the system is secure; further, on leaving the program the table will still be in such a state. (The specification does not state anything about the table during execution of the program.)

```
IN system_is_secure(privilege_table);
OUT system_is_secure(privilege_table);
```

For functions, return values can be specified. That is, when the subprogram returns normally, the returned value will satisfy certain conditions. The short form of RETURN annotation gives the value that is returned. A longer form is available that allows the user to place conditions on the return value without explicitly stating what it is. For example, a system may contain an access checking function, which given a user_id and item finds out whether the user has access to the item. We may write this in either of the following ways.

[2] This language, sometimes called Ada′, is observationally equivalent to a subset of Ada under certain assumptions. For example, program constructs that depend on the representation of data are excluded, and it is assumed that the *storage error* exception is not raised.

```
RETURN is_allowed_access(user_id, item);
RETURN z SUCH THAT z=is_allowed_access(user_id, item);
```

Propagation annotations are used to indicate under what conditions a subprogram may terminate by raising exceptions and to promise that certain conditions hold if an exception is raised. A propagation *constraint* states that an exception may be raised only under certain input conditions. In the first example below `invalid_item` is raised just when on entry to the subprogram the condition `user_is_secure(user_id)` is false. (Constraints are also available to state that an exception is raised if a condition holds on invocation, or only if the condition holds on invocation.) A propagation *promise* states that if the subprogram raises an exception then a certain predicate is guaranteed to hold. In the second example, even if `invalid_item` is raised, `system_is_secure(privilege_table)` will still be true.

```
RAISE invalid_item ⇔ (NOT (user_is_secure(user_id)));
RAISE invalid_item ⇒
             PROMISE system_is_secure(privilege_table);
```

The user may also introduce *embedded assertions*, which may be *cut point* assertions or *simple* embedded assertions. Assertions are not part of the specification of a program, but help the user to pinpoint errors by making claims about what is true at a certain point in the program. A simple embedded assertion makes a partial claim, which is incorporated into the precondition of the following part of the program. Thus, asserting P at a certain point in the program claims that P is true in addition to whatever else may hold at that point in the program.

Cut point assertions represent a more complete claim, in that the cut point assertion must imply the precondition of following part of the program, and becomes the postcondition of the preceding part. Thus a cut point assertion breaks a program up into two smaller pieces, each of which may be verified separately. The smaller pieces are presumably easier to verify, and in particular it should be easier to isolate errors: for example, the user may find that what he believes should hold at a certain point of the program cannot in fact be proven to hold.

Function names appearing in annotations, such as `is_allowed_access`, are names of mathematical functions defined on the mathematical objects underlying the Ada objects in the program. They do *not* refer to Ada functions, whose meaning depends on the semantics of execution and which are defined on Ada objects rather than mathematical domains. The mathematical functions are used to describe the semantics of the Ada functions. Such functions are defined by giving their signatures and a set of axioms in the form of rewrite rules in the Larch Shared Language.[3] For example,

```
INTRODUCES max:  Int, Int → Int;
AXIOMS:
     max(m, n) = IF n > m THEN n ELSE m;
END AXIOMS:
```

The user can also state various lemmas about the functions that follow from the axioms. Such lemmas are useful in proving verification conditions. Of course, extending the language in this way introduces the possibility of creating an inconsistent or unsound theory. Researchers at MIT and DEC SRC are developing several tools [8,9] that enforce language restrictions ensuring that this does not occur.

Users need to employ the Larch Shared Language to define new function symbols and also to define the mathematics associated with private types. An Ada private type is actually two types: an implementation type whose semantics follows from the semantics for Ada's base types and type constructors, and an abstract type whose semantics the user needs to supply. He does this by introducing Larch functions for the abstract type. For example, a private type Stack would have functions push, pop, top, etc. defined for it by means of appropriate rewrite rules.

3 Developing a Program

Penelope is used interactively to verify that a program meets its specification. By inspecting the verification conditions (VCs) computed from the specifications and text, the user learns about his program. If the VCs are provable, the program is correct. If not, the user analyzes the VCs to find what conditions he has neglected or incorrectly treated, using this information to add or modify text. Penelope recomputes all the VCs after each change the user makes to the text, and the cycle repeats. In the examples below, we illustrate the basics of programming with Penelope.

[3] In the current version of the editor a simplified subset of the Larch Shared Language is used.

3.4 A simple example

We will verify an Ada function ArrayMax that computes the largest value in an array. That is, given an array A and upper bound n, it returns the largest value of A(j) for index j ranging from 0 to n − 1. Our first step is to edit the specified stub for the function seen in Figure 1. We develop the Ada and embed the specifications as Ada comments; from the text and specifications, Penelope computes and displays the preconditions and VCs.

Figure 1:

```
FUNCTION ArrayMax(A : IN intarray; n : IN integer) RETURN integer
  --| WHERE
  --|      IN (n > 0);
  --|      RETURN maximum(A, n);
  --| END WHERE;
  --! VC Status: ** not proved **
  --! 1. n > 0
  --! >> m = maximum(A, n)
  --! <proof>

IS
  m : integer;

BEGIN
  <statement>
  --: PRECONDITION = (m = maximum(A, n));
  RETURN  m;
END ArrayMax;
```

The precondition of the Ada RETURN statement states that at the time control reaches that statement, the value of the Ada variable m must be maximum(A,n) We use the same name 'A' for both the Ada array and the Larch/Ada object corresponding to it. This is a harmless pun, since in our semantics, the Larch/Ada name denotes the value contained by the Ada object. Likewise, 'n' does double duty.

We do not use the name 'ArrayMax' to represent the Ada function in Larch/Ada annotations, since Ada functions are processes, not objects, and do not contain values. Instead, we specify the value computed by ArrayMax by means of the RETURN annotation, to be the value of the mathematical function maximum applied to the mathematical objects A and n.

The function maximum is defined elsewhere in a Larch/Ada trait by the two axioms:

```
maximum(A, 1) = A[0]

(k > 0) -> maximum(A, k+1) =
           if A[k] > maximum(A, k)
             then A[k]
             else maximum(A, k)
```

which together say that maximum(A, n) is the greatest value in the collection {A[0], A[1], . . . , A[n − 1]}. Notice that we use square brackets with Larch/Ada arrays instead of round brackets: the Larch/Ada object A[1] is the value contained in the Ada object A(1).

Displayed as comments in Ada text, VCs have this general form:

```
--! 1. Hypothesis
--! 2. Hypothesis

  ......

--! >> Conclusion
```

The hypotheses and the conclusion are mathematical statements about the program. When the conclusion follows logically from the hypotheses, the VC *holds*. An example of a VC that holds is

```
--! 1. n = 7
--! 2. f(n) = n*4
--! >> f(7) = 28
```

The VC for ArrayMax states that if the mathematical description in the IN condition holds on entry, then the value returned by the Ada function is mathematically described by the RETURN specification. Penelope calculated this VC by taking the IN condition as the hypothesis, and obtaining the conclusion from predicate transformation of the RETURN condition through the Ada text. Since predicate transformation depends on the text, the VC will be recomputed every time the text changes.

At the time that the user enters a correct proof of the current VC into the place now held by

```
--! <proof>
```

the status line will change to show that the VC has been proved. We will discuss proof elsewhere; for now we note that a correct proof will be possible only when the program is complete and meets its specifications.

Notice that while both specifications and VCs have the form of Ada comments, specifications are indicated by the prefix

```
--|
```

whereas VCs are indicated by the prefix

```
--!
```

Notice also that like the proof placeholder above, the statement placeholder

```
<statement>
```

indicates where Penelope permits further development. This development should obviously be of a loop to step through the array while computing the running maximum.

3.5 Guided programming

The techniques we demonstrate in programming the loop of **ArrayMax** are more powerful than is required for such a simple loop, but the simplicity of the loop makes it a good showcase for those techniques. In Figure 2, we show the loop very nearly complete, with the loop VC suggesting the completion.

Penelope generates a new VC for the loop, distinct from the main VC for the function. Both must be proved for the program to be verified. The loop VC is generated from the user's specification of the ***loop invariant*** (a generalization of the loop postcondition about which we will say more later), from the previously calculated loop postcondition (the precondition of the RETURN), and from the text of the loop body.

The user decided to calculate the maximum by finding the running maximum from 0 to j, as j runs from 1 to n, using a **WHILE** loop to do so. At present, the sole effect of the loop body is to increment the loop index j. The VC of the loop reflects that the loop is incomplete.

Figure 2:

```
  --! VC Status: ** not proved **
  --! 1. 0 < j
  --! 2. m = maximum(A, j)
  --! 3. j < n
  --! >> m >= A[j]
  --! <proof>

  WHILE j < n LOOP
    --| INVARIANT = ((0 < j) AND (j <= n) AND (m = maximum(A, j)));
    <statement>
    j := j+1;
  END LOOP;
  --: PRECONDITION = (m = maximum(A, n));
  RETURN  m;
END ArrayMax;
```

Hypothesis 1 of the loop VC gives the lower bound for the loop index. Hypothesis 2 of the loop VC is the assumption that at entry to the loop body, m is the running maximum for indices less than j. Hypothesis 3 states that the loop body will be executed. The conclusion is that m is at least as large as A[j], the next value of the array to be treated. Since this does not hold in general, the VC cannot be proved.

The loop VC suggests that the user include an IF statement to change the value of m so as to make the conclusion true. The user replaces the statement placeholder with the Ada code

```
IF A(j) > m THEN
  m := A(j);
END IF;
```

whereupon Penelope interactively recomputes the loop VC to be

```
--! 1. 0 < j
--! 2. m = maximum(A, j)
--! 3. j < n
--! >> IF m < A[j]
--!      THEN A[j] = maximum(A, j+1)
--!      ELSE m = maximum(A, j+1)
```

which is provable. The loop is complete.

3.6 Correcting mistakes

In the above, we showed how the user can be guided in adding to a program. Now let us see how a VC can help him in detecting and correcting errors.

Figure 3:

```
  --! VC Status: ** not proved **
  --! 1. 0 < n
  --! >> A[1] = maximum(A, 1)
  --! <proof>

IS
  j : integer := 1;
  m : integer := A(1);

BEGIN
  WHILE j < n LOOP
```

In Figure 3, a plausible common error has been made in initializing the running maximum before entering the loop. The initialization of the running maximum m should not be to A(1), but to A(0), since maximum(A, j) is the largest value of A[k] for $0 \leq k < j$.

The displayed VC is the main VC for ArrayMax. Its conclusion does not follow, since the right hand side reduces to A[0], and it is not true in general that A[1]=A[0]. Recognizing this (perhaps after a failed proof attempt), the user changes the initialization to read

```
  m : integer := A(0);
```

after which the VC is recomputed to the provable

```
  --! 1. 0 < n
  --! >> A[0] = maximum(A, 1)
```

3.7 Loops and invariants

In order to compute VCs, Penelope first computes a precondition for each Ada statement, a formula which must be true at the time that control reaches that statement. The main VC of the program states that if the `IN` conditions hold, then the precondition of the program holds. For a `WHILE` loop, the loop VC states that execution of the body preserves the *loop invariant.*

The invariant is the bridge across the loop. If it holds when the loop is first entered, if it is preserved by the body until termination, and if it implies the loop postcondition upon termination, then the loop is correct: it transforms the state described by the loop's precondition into the one described by its postcondition.

Choosing an invariant is an art beyond the scope of this paper (the interested reader is referred to the discussion in Gries [11]). In general, however, good invariants have these important features: they generalize the postcondition, and they encapsulate the essential conditions true within the loop. This can be clearly seen in Figure 4 in the invariant of the loop of `ArrayMax`.

Figure 4:

```
  WHILE j < n LOOP
    --| INVARIANT = ((0 < j) AND (j <= n) AND (m = maximum(A, j)));
    IF A(j) > m THEN
      m := A(j);
    END IF;
    j := j+1;
  END LOOP;
  --: PRECONDITION = (m = maximum(A, n));
  RETURN m;
END ArrayMax;
```

The first condition in the invariant, `(0<j)`, ensures a sensible meaning for the term `maximum(A, j)`. Also, together with the condition `(j <=n)` it expresses that the index `j` is ranging over the values that must be considered. The condition `(m=maximum(A, j))` generalizes the postcondition, stating that the running maximum has been correctly computed so far.

All these conditions are true upon first entry to the loop. All are preserved by the execution of the body. When the loop terminates, the additional condition `(j >=n)` holds. From this and the invariant, the postcondition follows.

3.8 Exceptions

In Penelope, one has the choice of assuming in an IN annotation that a forbidden condition will not occur, or stating in a propagation constraint that an exception will be raised if it does. If IN annotations are used, it is the responsibility of the caller to ensure that the IN conditions hold at call time. The precondition of the call in the calling program will include the IN conditions of the called program, with the actual parameters substituted for the formals. If propagation annotations are used, it is the responsibility of the called program to ensure that exceptional conditions are correctly treated. Clauses are added to the postcondition of the called program which ensure that its VCs cannot be satisfied unless the appropriate Ada RAISE statements are employed, as we will see below. Either way, the program can be verified to behave correctly given correct inputs. The VCs of the called program are similar, but not identical, in the two cases.

A variation on the program above written using exceptions is shown in Figure 5. The IN condition (0 < n) is removed, and no longer appears as a hypothesis of the main VC. Instead, the conjunct (NOT (n <= 0)) arising from negation of the propagation condition is added to the postcondition of the program. It persists implicitly in the precondition of the loop, following from (0 < j) AND (j <= n). Without the RAISE statement, the main VC would be

```
--! >> (0 < n) AND (A[0] = maximum(A, 1))
```

which cannot be proved, since there is no hypothesis on n. However, we see that with the RAISE statement, the VC is transformed to

```
--! >> (n <= 0) OR (A[0] = maximum(A, 1))
```

which states that either n is such that an exception will be raised, or the initial conditions for the loop are correct.

Figure 5:

```
FUNCTION ArrayMax(A : IN intarray; n : IN integer) RETURN integer
  --| WHERE
  --|      RETURN maximum(A, n);
  --|      RAISE invalid_array_length <=> IN (n <= 0);
  --| END WHERE;
  --! VC Status: ** not proved **
  --! >> (n <= 0) OR (A[0] = maximum(A, 1))
  --! <proof>

IS
  j : integer  :=  1;
  m : integer  :=  A(0);

BEGIN
  IF n <= 0 THEN
    RAISE invalid_array_length;
  END IF;
  --: PRECONDITION = ((0 < j) AND (j <= n) AND (m = maximum(A, j)));

      ......

  WHILE j < n LOOP
    --| INVARIANT = ((0 < j) AND (j <= n) AND (m = maximum(A, j)));
    IF A(j) > m THEN
      m := A(j);
    END IF;
    j := j+1;
  END LOOP;
  --: PRECONDITION = ((m = maximum(A, n)) AND (NOT (n <= 0)));
  RETURN  m;
END ArrayMax;
```

3.9 Cut point assertions

It can be seen from looking back on these figures that the verification of the program has effectively been distributed between showing in the main VC that the initial conditions for the loop are correct, and showing in the loop VC that the loop is correct. This incremental capability to separate concerns helps the user to isolate possible errors in his program and hence makes Penelope a powerful tool for program verification.

Another way to distribute verification is to use a cut point assertion to break the program into logically connected blocks to be independently verified. The precondition of the cut point becomes the cut point assertion itself, and the VCs of the first block are computed relative to it. A new cut point VC is generated, stating that the cut point assertion implies the precondition of the second block. Effectively, the cut point assertion becomes the OUT condition of the first block and the IN condition of the second.

3.10 Programming in the style of Gries and Dijkstra

The characteristics of Penelope that we have been describing make it an excellent tool for program development in the style advocated by Gries [11] and Dijkstra [5]. In this style, the programmer begins a program by examining the postcondition of the program and tries to develop a statement such that the precondition of the statement will more closely approximate the entry condition of the program. When he has written such a statement he examines *its* precondition to develop the penultimate statement of the program. At each step, as the programmer is working his way backward through the program, he is guided by the syntax and content of the precondition he is examining. Programs developed in this way are developed to meet a specification; when the programmer completes his program by writing the topmost statement, the precondition of that statement is just the entry condition of the program.

Penelope lends itself to programming following this methodology, because the verification conditions are generated by computing the preconditions of Ada statements in just this way. By automatically computing preconditions and verification conditions, Penelope makes it feasible to apply Gries's methodology to the Ada language.

3.11 Summary

The user employs loops and cut point assertions to subdivide the program into several regions, each with its own VCs. Within each region his progress is guided by automatic interactive recomputation of these VCs. Since the sum of their complexities is less than the complexity of the VC for the monolithic case, the distribution of verification across many program results in a significantly easier task for the user.

4 Proving Verification Conditions

Given a specification and a program, Penelope produces verification conditions; if the verification conditions can be proved, then the program has been shown to meet its specification.[4]

The natural approach to proving a verification condition in Penelope is to formulate lemmas containing all of the mathematical content of the verification condition. The remainder of the content is program-specific. The lemmas may be proved within the context of Larch,[5] and are available during proof of the verification conditions.

Penelope includes a simple proof editor that enables the user to instantiate axioms and lemmas, simplify the verification condition using an external simplifier, and give a "manual assist" to simplification if necessary. Associated with each verification condition in Penelope is a proof that takes its hypotheses and goal from the verification condition. The user constructs a proof tree by steps of the form "apply a given axiom," "apply a given lemma" or "simplify." It is also possible to use proof steps based on the syntax of the hypotheses or goal.

Penelope enables the user to communicate with an external simplifier. Currently we are using the State Delta Verification System (SDVS) [18] implementation of the Nelson-Oppen method for combining decision procedures [14]. We are using the (partial) decision procedure for the theory of integers under addition that is a component of the SDVS simplifier, and we are writing decision procedures for arrays and records that are compatible

[4] This was shown formally by Guaspari [12].

[5] In the future the Larch checker will be available for proving the lemmas. Currently, the proof of the lemmas is outside the scope of th : editor.

with the theories of Ada arrays and records used in Penelope. Completion of these decision procedures should make the proof of verification conditions considerably easier, since the SDVS simplifier is able to exploit knowledge about equalities in multiple domains. For example, it can simplify the expression

```
x = 0 AND a(x) = y AND NOT (a(0) = y)
```

to FALSE. In practice, appeal to axioms and lemmas combined with use of the simplifier suffices to prove many verification conditions.

The proof editor also enables the user to intervene "manually" to direct simplification. This is done via deduction rules permitting the substitution of equals for equals, etc. In particular, quantified formulae are not simplified by the external simplifier; for them the user must apply deduction rules using the proof editor.

Penelope also enables the user to insert in the program text directives to invoke the simplifier and to instantiate axioms and lemmas. This distributes the verification effort throughout the program. In Figure 5, the displayed VC is the result of several proof steps and simplifications, which were suppressed as extraneous to the exposition. Here, we concentrate on those steps. In Figure 6, we show how to apply directives to the precondition of the first IF statement so as to reduce the main VC to a triviality.

Reading up from the IF statement, we see several guided predicate transformations. The first step is to embed a simple assertion expressing conditions on n, m and A that are true at the time control reaches that statement, thus making these facts available to the simplifier. The first two conditions are self-explanatory; the third states that n has the value at the point of the assertion that it had on entry to the subprogram. The transformation the embedded assertion effects is to conjoin these conditions to the precondition.

The SIMPLIFIED PRECONDITION transformation next invokes the external simplifier to exploit the additional information just asserted. The result (not shown) contains the term maximum(A, 1), which is then transformed by application of the axiom governing the base case into A[0]. The result of a second simplification, exploiting this substitution, is the first precondition shown. This precondition reduces to TRUE by transformation through the declarations.

The effect of such steps as these is to factor the proofs of verification conditions so that after predicate transformation there may typically be little

Figure 6:

```
FUNCTION ArrayMax(A : IN intarray; n : IN integer) RETURN integer
  --| WHERE
  --|      RETURN maximum(A, n);
  --|      RAISE invalid_array_length <=> IN (n <= 0);
  --| END WHERE;
  --! VC Status: proved
  --! BY synthesis of TRUE

IS
  j : integer := 1;
  m : integer := A(0);

BEGIN
  --: PRECONDITION = ((j = 1) AND (m = A[0]) AND (n = IN n));
  --: SIMPLIFIED PRECONDITION;
  --: USE AXIOM m1 IN TRAIT T WITH Arr = A;
  --: SIMPLIFIED PRECONDITION;
  --| ((j = 1) AND (m = A[0]) AND (n = IN n));
  --: PRECONDITION =
      (IF (n <= 0)
        THEN IN (n <= 0)
        ELSE ((0 < j) AND (j <= n) AND (m = maximum(A, j))));
  IF (n <= 0) THEN
    RAISE invalid_array_length;
  END IF;
  --: PRECONDITION = ((0 < j) AND (j <= n) AND (m = maximum(A, j)));
```

or nothing left to prove. This is the approach used (in a less formal context) by Gries [11]. Gries assumes an intelligent human being performing predicate transformation, and implicitly simplifying, applying mathematical theorems, etc. Verification conditions for programs in Gries's methodology are usually completely trivial: the precondition of the program is exactly the input condition.

5 Related Work

In this section we compare Penelope with some well-known verification and specification systems.

Gypsy The Gypsy Verification Environment [10] is probably the best known automated system for formal verification. The Gypsy programming language is a Pascal-like language, together with versions of data abstraction, exception-handling, and asynchronous concurrency (through shared buffers).

Gypsy associates VCs only with whole programs. Gypsy's VC generator, however, breaks a program into paths and, essentially, associates a VC with each path. This helps the user to isolate errors. Further, suppose that the VC associated with path p has been proven, that the program is subsequently modified (perhaps because VCs associated with other paths cannot be proven), and that the new VC generated for path p is identical with the old one: in this case the VC need not be reproven.

Effort has recently been invested toward making Gypsy a production tool. Penelope possesses no comparable sophistication in theorem-proving, library facilities, etc.

Anna Anna [7] is a specification language for Ada and, as indicated above, has served as a model for much of Larch/Ada. The Anna project is an effort to introduce specification to Ada programmers by providing specification constructs which can be checked at runtime. The semantics of Anna annotations is computational rather than mathematical.

EVES The underlying programming language is a simple sequential language (called m-Verdi) [3]. Like the Gypsy language, m-Verdi is designed for verification, and therefore the mathematical apparatus underlying EVES is much simpler than that underlying Penelope. Much effort has been devoted to the engineering of a theorem prover: using the Nelson-Oppen algorithm [14] (from the Stanford Pascal Verifier [6]), heuristics from the Boyer-Moore prover [2], etc. The principal difference in "spirit" between EVES and Penelope is our emphasis on incrementality.

AVA Computational Logic, Incorporated is working on AVA (A Verifiable Ada [4]). Their strategy is to define the semantics of an Ada subset in Boyer-Moore logic [2], and then to reason directly about the denotations of Ada programs in Boyer-Moore logic, without the intermediary of VCs. Currently, AVA has no distinct specification language. A program is specified by making the appropriate assertions about its denotation.

SDVS Aerospace Corporation is also applying a pre-existing tool, the State Delta Verification System [18] (like the EVES prover, implementing the Nelson-Oppen method), originally designed for verification of micro-code. They model the semantics of Ada constructs in terms of the "low level" semantics of SDVS and apply the SDVS prover to the result.

6 Future Work

Penelope is still under development. This section describes some extensions we plan to the theory and implementation of Penelope.

We intend to expand the subset of Ada covered by Penelope. We are currently in the process of adding private types to packages.

The computation of weakest preconditions is based on continuation semantics [15], which is well-suited to verifying partial correctness of sequential programs. We have briefly investigated ensuring total correctness, and we believe it is straightforward to verify total correctness in the absence of mutually recursive functions. We do not yet know which of several approaches is the best way to verify total correctness when mutually recursive functions are allowed. We are also investigating ways to extend a continuation semantics to address concurrency [16].

Because we rely heavily on the use of axioms and lemmas in proving verification conditions, we want to integrate the Larch checker into our system. This will allow the user to build up a library of theory on which he can draw in verifying new programs.

7 Conclusion

The interactive approach of Penelope is founded on three principles:

- the user can inspect (incomplete) verification conditions and use the insight gained in order to complete or correct his program;
- verification conditions are linked to syntactic constructs, which aids the user in finding and correcting programming errors;
- the ability to formulate the mathematical content of programs in axioms and lemmas, together with the availability of a powerful simplifier, allows verification conditions to be proved relatively easily;
- the proof of verification conditions can be distributed, and partitioned into a "mathematical part" (proof of lemmas about the specificational notions introduced in a Larch trait) and a "programming part" (in which those lemmas, together with a powerful simplifier, may be invoked as needed), supporting an intellectual style that is natural to a human user.

7.12 Acknowledgements

The authors are grateful to Norman Ramsey and David Guaspari for thoughtful criticism of earlier drafts of this paper.

The software for Penelope is generated in part by the Synthesizer Generator under license from Cornell University. We acknowledge Thomas Reps and Tim Teitelbaum for their role in its development.

References

[1] Odyssey Research Associates. A short introduction to Larch/Ada-88. Odyssey Research Associates internal document, 1988.

[2] R. Boyer and J. Moore. *A Computational Logic.* Academic Press, 1979.

[3] Dan Craigen. A description of m-Verdi. Technical Report TR-86-5420-02, I. P. Sharp Associates, Ltd., 1986.

[4] Dan Craigen, Mark Saaltink, and Michael K. Smith. The nanoAVA definition. Technical Report 21, Computational Logic, Inc., Austin, Texas, 1988.

[5] Edsger W. Dijsktia. *The Discipline of Programming.* Prentice-Hall, 1976.

[6] D. C. Luckham *et al.* Stanford Pascal Verifier user manual. Technical Report STAN-CS-79-731, Stanford University, March 1979.

[7] D. C. Luckham *et al.* Anna: A language for annotating Ada programs. Technical Report CSL-84-261, Stanford University, 1986. Reference Manual.

[8] S. Garland. Private communication. Larch Checker.

[9] S. Garland and J. V. Guttag. An overview of LP, the Larch prover. In *Third International Conference on Rewriting Techniques and Applications*, 1988.

[10] D. I. Good, R. L. Akers, and L. M. Smith. Report on Gypsy 2.05. Technical report, Computational Logic Inc., 1986.

[11] David Gries. *The Science of Programming.* Springer-Verlag, 1981.

[12] David Guaspari. Formal definition of satisfaction. Odyssey Research Associates internal document, 1988.

[13] J. V. Guttag, J. J. Horning, and J. M. Wing. Larch in five easy pieces. Technical Report TR 5, DEC/SRC, July 1985.

[14] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.

[15] Wolfgang Polak. Program verification based on denotational semantics. In *Conference Record of the Eighth Annual ACM Symposium on Principles of Programming Languages*, 1981.

[16] Wolfgang Polak. Formal verification of Ada tasking programs. Odyssey Research Associates internal document, 1988.

[17] Norman Ramsey. Developing formally verified Ada programs. In *Proceedings of the Fifth International Conference on Software Specification and Design*, May 1989.

[18] T. Redmond. Simplifier description. Technical Report ATR-86A (8554)-2, Aerospace, November 1987.

[19] Thomas W. Reps and Tim Teitelbaum. *The Synthesizer Generator: A System For Constructing Language-Based Editors.* Springer-Verlag, 1988.

[20] I. Takeuchi. Private communication to J. McCarthy.

[21] D. G. Weber. Beyond A1 using Ada code verification. Submitted to the Software Testing, Analysis and Verification Symposium III.

Adding CASE Technologies to Formal Verification*

J. V. A. Janeri, J. S. Barlas, L. L. Chang
2 February 1989

The MITRE Corporation
Trusted Computer Systems Department
Burlington Road, M/S B-325
Bedford, Massachusetts 01730

Abstract

We propose to use CASE technology to further automate the labor-intensive task of formal verification, by integrating the process of formal software design verification with the software engineering life cycle. Hypertext and graphics are two CASE features that would be especially useful for this integration. We plan to build a prototype Specification Browser, based on an off-the-shelf CASE tool, to serve as a verification aid. The browser will exploit both hypertext and graphics capabilities to assist verifiers and evaluators with the organization of verification evidence. Scenarios presented here show how an evaluator would use the Specification Browser to examine a software system's design and its documentation.

Introduction

Software design verification is the process by which one formally specifies and proves the correctness of a design using formal methods. It is often used in verifying security properties of an operating system design [1]. Although this process increases the level of assurance in the design's correctness (via an accumulation of *verification evidence*), it is a labor-intensive task. Furthermore, formal verification requires the experience of a person who is well-versed in formal logic and has an understanding of the system being verified.

It is required that formal specifications be developed by vendors and reviewed by a team of independent evaluators if the system is to attain the highest level of assurance. These specifications are written in a formal specification language, in contrast to their traditional English language counterparts, the requirements specifications. The specifications contain conditional logic assertions stating how the abstract machine shall behave along with security properties that collectively define "secureness". A series of proofs are developed to formally demonstrate that the specifications uphold the stated security properties. Mappings that relate the formal specifications to all relavant code modules are supplied as additional evidence that the proven properties apply at the lower (implementation) level. When examining large and complex formal specifications, verifiers and evaluators are burdened with keeping track of the relationships between all of the documentation, including the proofs and correspondence mappings.

The state-of-the-archaic method employed today has verification and evaluation teams wading through hundreds of pages of computer listings and using yellow Post-it® pads for labeling the various sections of text and transforms. This is clearly not the way to proceed. As verification continues to mature, larger and larger systems will be candidates for formal

* This work was supported by the National Computer Security Center C3, and the United States Navy.

evaluations, and the specifications and documentation that will be presented as support will grow proportionally. We are outgrowing our days with tinkering with toy specifications and entering the age where we must find a better method to understand and review the voluminous formal and informal specifications.

There are many deep problems in the field of verification, such as deciding on a particular design paradigm for specifying a design or determining an appropriate underlying system of logic to use for a verification system. Our goal is more modest: developing a solution to the class of problems that can be solved in the *near term* and could have a dramatic effect on the way in which the verification/evaluation process proceeds. Difficulties can be solved in areas such as integrating formal verification into the software life cycle, requirements tracing, specification-to-implementation mappings, configuration management, and the basic organization of large-scale documentation. To this end, we are designing a verification-aid called the Specification Browser (SB) that relies on intrinsic CASE features, primarily graphics and hypertext. The Specification Browser will be a prototype and will enjoy the novelty of interfacing with an existing verification system. The prototype will be one step towards bringing together development and verification aids into a single, unified environment.

Motivation: Organizing the Formal Evaluation Materials

From our experience as *evaluators* of secure computer systems, we believe there is an urgent need for a software tool that will reduce the burden of this task. By tying together the many common aspects of software engineering and design specification and verification into one sensible and coherent representation, such a tool could ease the frustration that now accompanies the *analysis* of large-scale documentation.

MITRE's charter for performing this research is grounded in the area of formally specifying, verifying, and evaluating "secure systems." Organizing the suite of documentation that is required by the government-standard Trusted Computer Security Evaluation Criteria (TCSEC) [2] and the military standard for defense system acquisition, MIL-STD 2167A [3], was the primary motivation for this tool (a fact that is underscored by the examples that appear in this paper). It is now clear, however, that application of such a tool would not be limited only to security-related and government-standard document suites, but to any company's corporate standards, guidelines, and design documentation.

Approach: Analyze CASE Tool Requirements

The question of "What designates or defines a CASE tool?", for our concerns, can be defined simply as:

> *A tool (or set of tools) that when used under the guidance of a particular methodology, enhances a software or systems engineer's ability to build a software system.*

We note that the vast majority of CASE tools rely on graphics and hypertext concepts to reach the primary goal of assisting in the specification, design, and implementation of software systems. For our near-term prototype, we will exploit both hypertext and graphic functionality.

Given that we wish to alter an existing off-the-shelf CASE tool as a basis for the browser, it is important that the CASE tool have an *open architecture:* i.e., that the CASE tool be designed in such a way that it can be configured to meet users' needs and to allow easy interface with another tool. No modification of source code is necessary when using a CASE tool with an open architecture. We have found that this sort of extensibility is built into a very limited number of CASE products today.

We have performed a survey on twenty-one existing CASE tools and have concluded that no single tool includes all the desired features for our prototype. However, two tools appear to be the most appropriate bases for the prototype construction. One of them provides a nice linking functionality on which we can build a hypertext capability; the other one is promising for its sophisticated graphics representation.

After investigating the work of many researchers, vendors, and theorists in the CASE arena, we have found that the general quality and overall functionality of the software is very admirable. However, the lack of a formal and well-founded basis for many of the most popular methodologies is apparent. CASE tools with a formal semantic basis would help produce software systems with a higher degree of assurance (consistency, soundness, and completeness); but if one attempts to combine a verification methodology with an incompatible CASE methodology, the degree of assurance could lessen. This is an area for further research. With the appropriate formal foundational work having been documented by the CASE vendor, one could theoretically bypass any such potential "semantic clash" and detect any likely inconsistencies early in the project. Another alternative is to support various methodologies with one tool.

The hardware system we have chosen for the prototype is the Sun Workstation. We are taking full advantage of the workstation's multiprocessing capability, resident windowing system, and mouse device interface. In addition, it is the only workstation that supports both verification environments and CASE tools.

In the next section, we lay the groundwork for understanding the problem at hand: the management and organization of two sizable and separate documentation (requirements) standards. It is important to understand how each of the required deliverables relates to one another.

Large-Scale Document Organization and Topology

In Figure 1, we present the security-related deliverables required by the TCSEC and Military Standard 2167A arranged as "vertical" columns. The most abstract elements are presented at the top, and the most detailed elements are at the bottom of each column. Explicit relationships have been defined at each step. On the left are the deliverables required by the TCSEC: the Formal Security Policy Model (MODEL), the Descriptive Top-Level Specification (DTLS), the Formal Top-Level Specification (FTLS), and the source code (the code is considered part of both the documentation suites [illegible]s shown only once). The Model[1] is specified as the highest level of abstraction and formally describes the system security policy in terms of mandatory and discretionary access of active entities (e.g., processes, users) towards passive data objects (e.g., files, directories). The DTLS and the FTLS share approximately the same information, but the DTLS is written in a natural language, and the FTLS is specified in a formal specification/verification language,

[1] The word *model* in this case means something a little different from the traditional usage, since here it is used in a very distilled sense, i.e., to refer to the modeling of security-relevant access mediation.

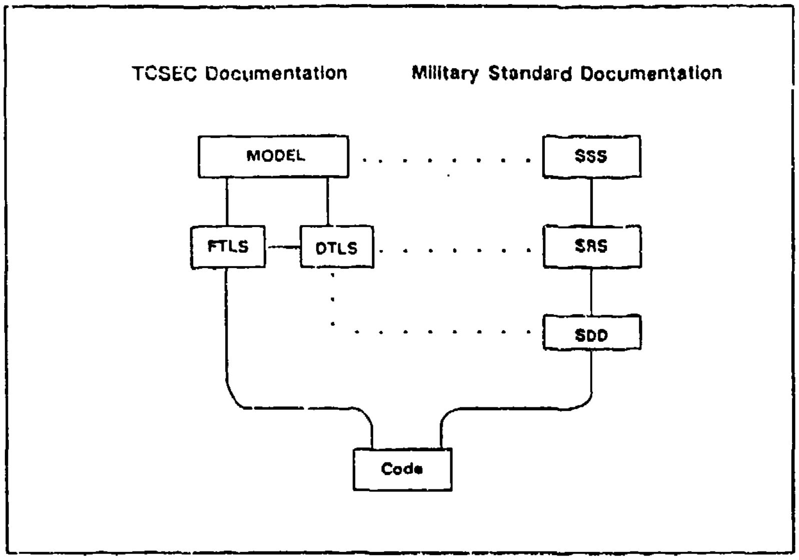


Figure 1. Parallel Organization of Software Documentation

much like first-order predicate logic.[2] The FTLS is formally proven to be consistent with the properties and rules of the model. These two representations describe the precise interface into the security-relevant software – more commonly referred to as the Trusted Computing Base (TCB). The code is the realization of the design and requirements specifications and is regarded as the lowest-level (implementation) of detail.[3] This decomposition follows the typical "top down" design methodology, where each successive lower level is a further refinement (and less abstract version) of its parent level.

Similarly, the Military Standard 2167A (Software Engineering Development Life cycle) deliverables are listed in parallel as: the System Segment Specification (SSS), the Software Requirements Specification (SRS), and the preliminary and detailed design versions of the Software Design Document (SDD). The 2167A is a detailed government standard imposed upon vendors and developers of defense systems. It covers the entire system development life cycle from early system specification and requirements definition, to design and implementation, through testing and maintenance. The linear organization in Figure 1 depicts the two suites as truly being parallels to one another; this is admittedly an oversimplification. By adding the dotted lines to represent the hypertext relationships, this high-level representation becomes arbitrarily close to the actual relationships between documents (and portions of documents).

Graphics and Hypertext Involvement

To speed up the system understanding process and to help evaluators perform their tasks more efficiently, it would be extremely helpful to connect formal documents required by the TCSEC to their corresponding MIL-STD 2167A requirements and design documents,

[2] In our case, we are using the formal specification language Ina Jo [4] which is part of the Formal Development Methodology (FDM) verification system [5].

[3] Certainly there are other levels below the code, but this project is focusing on the software aspects of the development life cycle. Future enhancements may address the other relevant microcode and hardware components.

and vice versa. For example, if an evaluator is trying to understand some Ina Jo construct, s/he could follow a pointer that connects the predicate logic sentences to some descriptive English text. This is a key feature of the Specification Browser; it is designed to provide a method for building and traversing "links" *within* each documentation suite and *between* documentation suites.

Hypertext can be used to affirm explicit relationships between differing entities. The links can be traversed, deleted, inserted, and modified, using an underlying database. The beauty of this concept is that the details of the database are hidden from the user, and the only contact a user has with data manipulation is via a bit mapped user interface, complete with windows, and a mouse.

Graphics also play an important role in increasing the ease of conveying information about the structural design of a system; they are what people turn to first when they would like to understand a large and complex system. Besides giving a high-level view of the system and increasing the sense of understanding the system structure, one can also create a graphical representation to form an active part of the user-interface. When the user would like to visit some section of the system, s/he positions the mouse-pointer on the appropriate icon and selects the entry (after which s/he is left to continue the analysis). This is accomplished by displaying a System Topology Diagram in a Sun window at the main menu level of the Specification Browser. Regardless of whether the system is arranged hierarchically, cyclicly, or linearly, the graphics can be used as a navigational aid to the various sections of the system. The Specification Browser exploits other qualities of graphics as well. Obviously, the graphics can be sent to a laser printer strictly for use as presentation material, and as the field of of Visual Formalisms matures, eventually the graphics may be used in formally specifying and verifying software designs and code. Some of the graphics that are produced as part of the software development activity can be used during verification and can help bring together the software engineering community and the verification community.

For each system, the structure would be different, but easily reprogrammable using a typical graphic capture algorithm. Also, the on-screen display of the chart would not be passive, but could be used as an interactive map of the system *topology*. To illustrate some of these concepts, a high-level view of the browser system is described below. It clearly shows how powerful a specification tool such as this would be—one step toward providing some real solutions to the large-scale documentation problem.

System Description

The Specification Browser can be invoked in two different manners, depending on whether the browser has been configured for the vendor's own use, or whether the browser is intended for the evaluator. The proposed additions to the existing interface will be designed such that the developer/vendor of the software product has both read and write access to the hypertext database, whereas the evaluators' version of the browser will be configured with read-only access to the hypertext database. The Read-Write Browser (RWB) is the software specially configured for the vendor. The evaluator will invoke another form of the tool, the Read Only Browser (ROB). This is the software that enables the user to traverse links specified within the hypertext database, without affecting the integrity of the database. The early prototype of this read-only version of Specification Browser is being designed with consideration for future versions that might enforce a discretionary access control mechanism, giving certain privileged individuals the benefit of *modifying* the hypertext links. Figure 2 illustrates a general view of the browser and how the proposed interface, the existing CASE tool, the Formal Development Methodology (FDM)

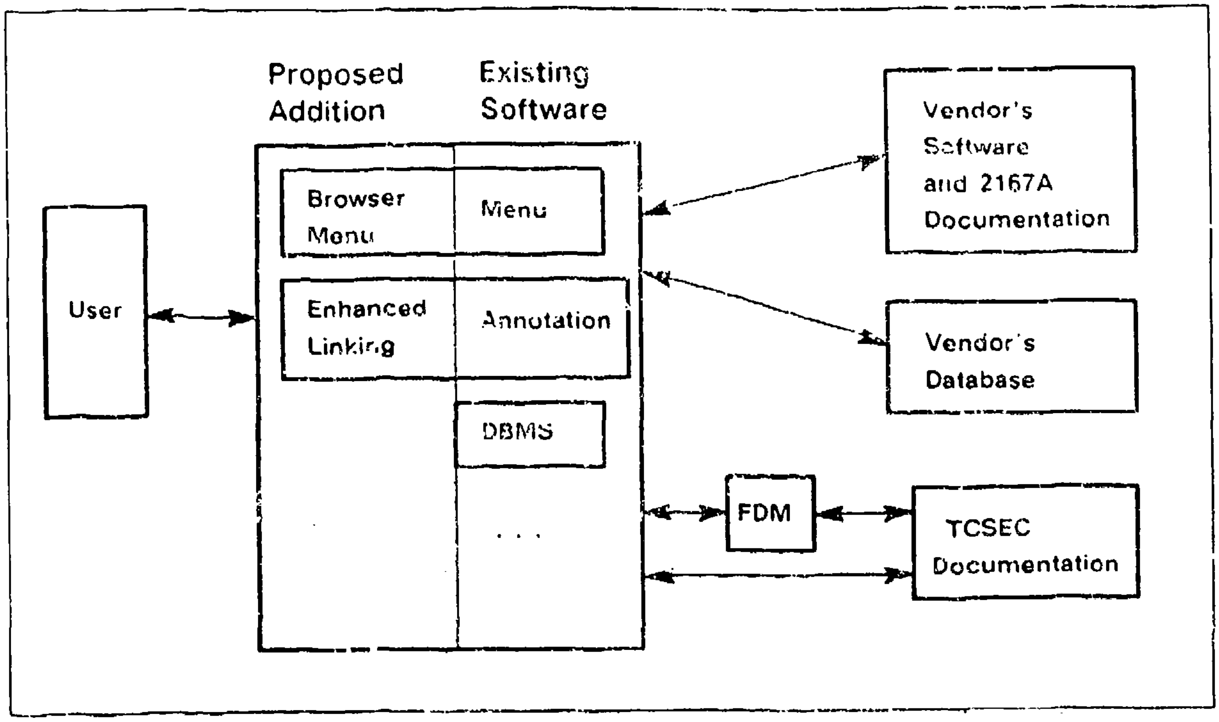


Figure 2. Browser System = User Interface + CASE

system, and the documentation relate to one another. The Specification Browser allows users to invoke the FDM system via the same user interface. In this context, a "user" of the browser system is an individual, either an evaluator or a vendor. The evaluator operates the browser only after the database of links has been put in place by the vendor.

The Specification Browser's Role

The Read-Write Browser is for inserting the initial links before the evaluation process begins. Since the vendor has intimate knowledge of the system and the relationships among its components (formal or otherwise), the vendor is the only candidate for inserting the links. An implied assumption here is that the vendor will be *required* to go through the entire system and *lace* documentation deliverables with code deliverables and formal deliverables.[4]

With each initial entry, a relational n-tuple is built and stored in the database. Each relation contains a time stamp that will be used by a configuration management system (CMS) and for future releases of the system being analyzed. The rationale is that the bindings between various entities will come and go, and the time stamp information is required to configure snapshots and proceed in the analysis during a changing (unfrozen) design.

[4] The process in which the vendor ties together entities from throughout the system deliverables is referred to here as lacing. The idea behind lacing is that the many representations of the system need to be related explicitly. It is reasonable to require some coherent relationship *among* the documentation to be supplied *as part* of the documentation.

When the evaluation is slated to start, the vendor delivers completed portions of their source code, TCSEC documentation, any required MIL-STD 2167A Documentation, and the relational/hypertext database to each of the evaluators. Upon arrival at the evaluator's site, these items are installed on a Sun Workstation under a user's account.

Evaluator's Scenario

As an example of how the ROB portion of the Specification Browser system works, suppose we are evaluating some network software, call it "Network X." We will assume that the vendor is responsible for both the formal work and the development of the software and has laced the components together to form an initial hypertext database. The formal work describes the network in terms of a massive collection of Ina Jo specifications, which must be organized into some manageable structure. The partitioning of the trusted computing base (TCB) across the major subsystems implies that the formal specifications are somehow divided up across the entire system.[5] Figure 3 shows the breakdown or the *topology* of the formal specifications for Network X based on the allocation of security-relevant features. This is an example display as the evaluator would see on his/her workstation monitor and is an example of use of graphics as an Active Graphic Interface. There are no limits (in theory) to the number of levels that can be specificed using the FDM system. In fact, there are formal specifications today that have been expressed in as many as seven (7) levels of Ina Jo specification. We will assume our imaginary Network system contains three levels of Ina Jo specification, in order to explain some of the Ina Jo specific features of the Specification Browser.

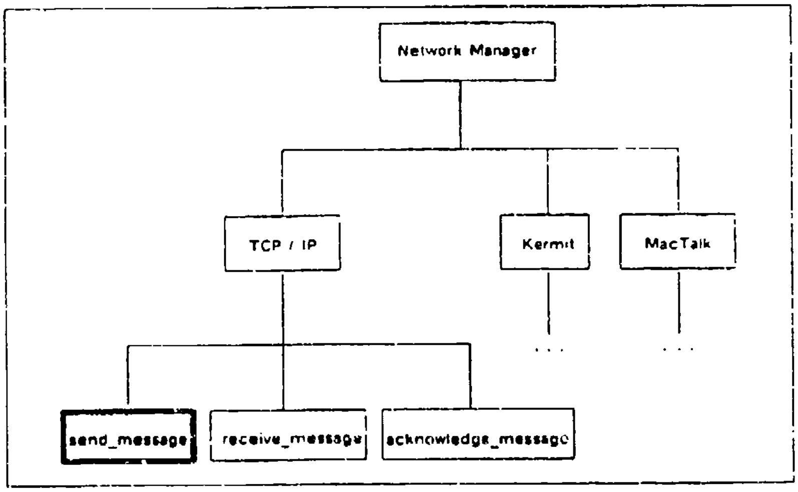


Figure 3. A System Topology Diagram for Network X

[5] The manner in which software is partitioned can vary drastically based on *what* the partitioning is emphasizing. Software systems engineers typically divide software up based upon functional requirements (or just "functionality"). Developers of security software follow a similar design paradigm, but emphasize security-relevant requirements rather than purely a functional decomposition. Caveat Lector: these two views are almost certainly never the same.

To begin the scenario, the user would position the mouse pointer on (or near) a node and select his/her area of interest. In Figure 3, the node that was selected is highlighted (SEND_MESSAGE). A window appears with the appropriate file(s) loaded into the window's buffer, and the evaluator is free to begin perusing the contents. The original graph is not lost, but turns into a system's Topology Icon and migrates to the upper border of the workstation screen. This non-intrusive icon organization is a popular technique used by the SunTools (tm) product (Sun Microsystems) and is a very useful short-term method for tracking opened files.

Inter/Intra Document Traversal

A series of different pull-down menus is supplied to the evaluator so s/he can traverse the system in many different contexts. The menu types available to the evaluator are shown in Figure 4.

Continuing with our scenario, Figure 5 shows that the evaluator is now looking at a particular FTLS transform for Network X. In Figure 6, the evaluator instructs the system to load the DTLS–the English language version of SEND_MESSAGE–into a new window shown as Figure 7. This is achieved via selecting the pull-down menu option DTLS and the transform SEND_MESSAGE with a 'click' of the Sun mouse. Then, one by one, the text files that were linked to the selected transform appear in a new window. If there is more than one DTLS text file associated with FTLS transform SEND_MESSAGE, one of the two possible alternatives can be chosen:

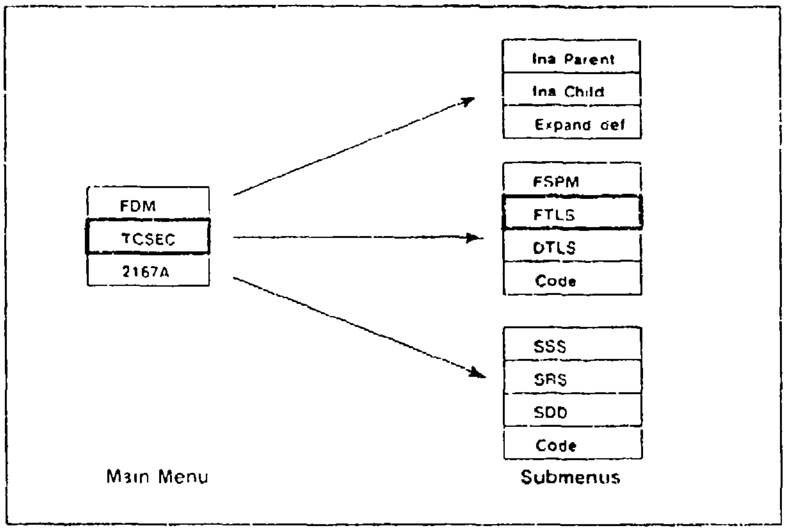
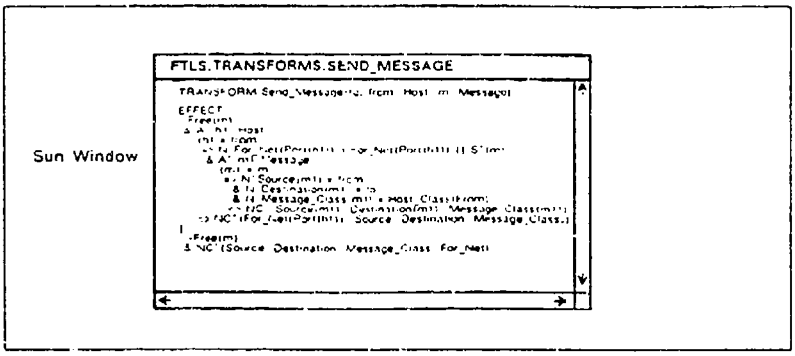


Figure 4. Pull- Down Menus of the Specification Browser

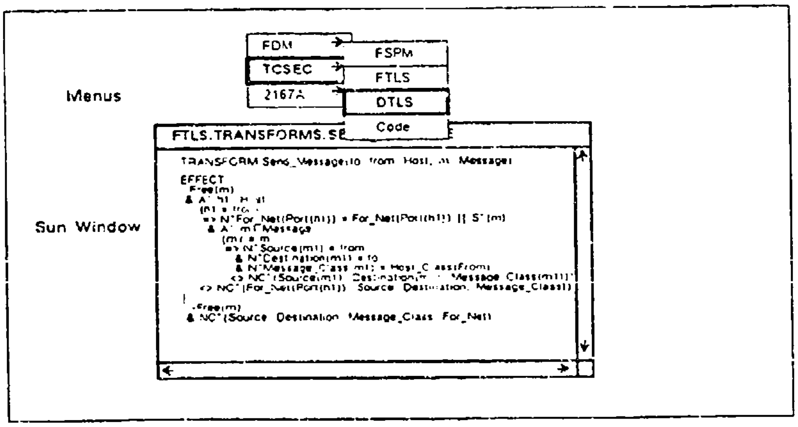
1) Only the first paragraph in the linked list will be displayed (and in a window of the appropriate size). Each successive paragraph is read into a newly created window.

2) If more than one entity is contained in the reference link, then a window is created to display the *members* of this chain. The user may then select one or more of the items (via clicking the mouse on pre-illustrated boxes) and upon leaving the selection window, the selected files will be piped into a Unix-like "more" utility.



```
TRANSFORM Send_Message(to from Host m Message)

EFFECT
 Free(m)
 & A' h1 Host
  (h1 = from
   => N'For_Net(Port(h1)) = For_Net(Port(h1)) || S'(m)
    & A' m1 Message
     (m1 = m
      => N'Source(m1) = from
        & N'Destination(m1) = to
        & N'Message_Class(m1) = Host_Class(from)
       <> NC'(Source(m1) Destination(m1) Message_Class(m1))
  <> NC'(For_Net(Port(h1)) Source Destination Message_Class))
|
 -Free(m)
 & NC'(Source Destination Message_Class For_Net)
```

Figure 5. The FTLS SEND_MESSAGE transform of Network X



```
TRANSFORM Send_Message(to from Host, m Message)

EFFECT
 Free(m)
 & A' h1 Host
  (h1 = from
   => N'For_Net(Port(h1)) = For_Net(Port(h1)) || S'(m)
    & A' m1 Message
     (m1 = m
      => N'Source(m1) = from
        & N'Destination(m1) = to
        & N'Message_Class(m1) = Host_Class(from)
       <> NC'(Source(m1) Destination(m1) Message Class(m1))'
  <> NC'(For_Net(Port(h1)) Source Destination Message_Class))
|
 -Free(m)
 & NC'(Source Destination Message_Class For_Net)
```

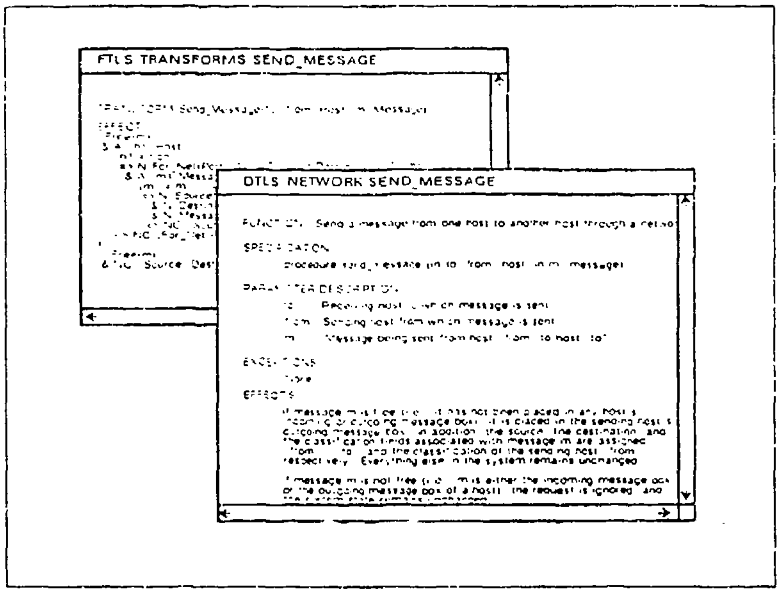
Figure 6. Selecting the DTLS text for SEND_MESSAGE

If an attempt is made to traverse between documents or within a document, and no associated correspondences are found in the database, then you are notified with the appropriate message.

We assume that if no pull-down menu item was selected, we are automatically going to browse files of the same type (i.e., other FTLS files) or remaining portions of the current file, i.e., other FTLS transforms within the current file. To change the context in which one is currently working, a pull-down menu item must be selected.

If the evaluator now chooses to view the corresponding CODE for this particular function, the same process is repeated, except the CODE pull-down menu option is selected (refer to Figure 4). A new window would appear and the program would begin loading in the computer code (see Figure 8).

If the pull-down menu option INA PARENT had been chosen when looking at the FTLS, the user could then click on SEND_MESSAGE, and the upper-level transform (in our example, this is TCP/IP) would appear in a newly created window. As long as one reselects the INA PARENT menu option, any entity that is selected in the main window will be subject to further ancestral searches.

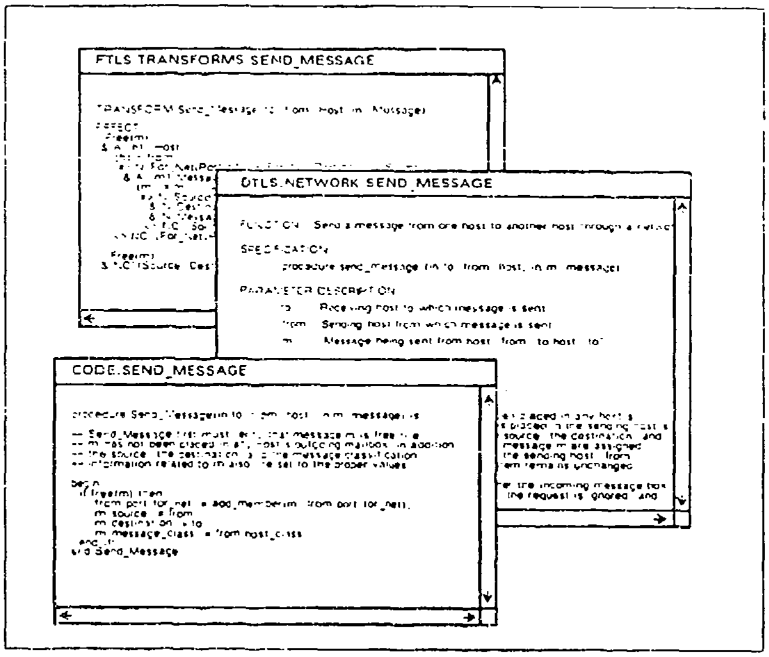


FTLS TRANSFORMS SEND_MESSAGE

DTLS NETWORK SEND_MESSAGE

```
FUNCTION: Send a message from one host to another host through a netwo

SPECIFICATION
      procedure send_message (in to, from: host; in m: message)

PARAMETER DESCRIPTION
      to      Receiving host to which message is sent
      from    Sending host from which message is sent
      m       Message being sent from host "from" to host "to"

EXCEPTIONS
      None

EFFECTS
      If message m is free (i.e. it has not been placed in any host's
      incoming or outgoing message box), it is placed in the sending host's
      outgoing message box. In addition, the source, the destination, and
      the classification fields associated with message m are assigned
      "from", "to", and the classification of the sending host "from"
      respectively. Everything else in the system remains unchanged.

      If message m is not free (i.e., m is either the incoming message box
      or the outgoing message box of a host), the request is ignored, and
```

Figure 7. New foreground window with description of SEND_MESSAGE



FTLS TRANSFORMS SEND_MESSAGE

DTLS NETWORK SEND_MESSAGE

CODE.SEND_MESSAGE

```
procedure Send_Message(in to, from: host; in m: message) is

-- Send_Message first must verify that message m is free, i.e.
-- m has not been placed in any host's outgoing mailbox. In addition
-- the source, the destination, and the message classification
-- information related to m also be set to the proper values

begin
  if free(m) then
      from.port_for_net := add_member(m, from.port_for_net);
      m.source := from;
      m.destination := to;
      m.message_class := from.host_class;
  end if;
end Send_Message
```

Figure 8. Three simultaneous representations of SEND_MESS.\GE

Besides enabling one to traverse various Ina Jo components (variables, types, definitions, transforms, etc.) at different levels of abstraction, there is a menu item EXPAND DEF which pops up a separate window and displays the right-hand side of the selected definition (definiens). This micro-level[6] intra-document binding is extremely useful during the evaluation and theorem proving stages of verification. Both Parent-level and

[6] Compare with the notion of macro-level intra-document binding which is the power behind supporting requirements traceability.

Child-level terms are displayed in a pop-up window with a predefined format, and an option to generate an automatic report is supplied.

Transform Report

Child Transform:	SEND_MESSAGE
Child Level:	2LS
Parent Transform:	TCP/IP
Parent Level:	TLS
Mapping:	TCP/IP == SEND_MESSAGE

If the evaluator ventures across several links (and possibly several Ina Jo levels), s/he could get lost in the system. This is no problem. At any point, reselection of the saved System Topology Icon would enable the user to recall instantly where s/he is in relationship to the rest of the system. Finally, if the user no longer requires the chart, the mouse pointer is moved to the icon marquise, and selects a pull-down menu to close the chart ... with this, the icon vanishes.

Once the evaluation begins, and each member is assigned his or her own section to concentrate on, little will change with respect to the evaluator's working patterns. Therefore, a profile of the user's common activities would be stored in a local file and accessed upon entry into the system. The System Topology Graph, which appears automatically at initialization of the browser, would be shared by all users.

Notecards

Although evaluators are only able to read files and links, they are able to write to notecards associated with files. Each user can have one notecards associated with each file he has access to. All the notecards associated with a particular user are implemented as local files that are only accessible by this user. Comments can be inserted in these notecards by both evaluators and developers/vendors. To insert or modify the content of a notecard, a user first must select "notecard" on the menu, which opens a window containing the current version of the notecard associated with the file in the active window. After the note window has been opened, the user can edit the content of this window.

The notecard feature is a very nice additional place to put comments, where they are on-line, accessible, and always correctly associated with what they are commenting on. Without such a capability, notecards and other non-on-line note taking would have to be typed in to be included in reports or electronic mail messages. This feature would be helpful to include in the design and could easily be ported to other (Sun-based) tools as an add on editor option. It is also a nice alternative to in-line comments and systems such as the one proposed by Knuth in [6].

In addition to the notecard capability, a report containing information stored in these notecards also can be generated by the selection of the "note_report" menu item. One capability that we eventually would like to implement is to permit a user to specify the scope of the information that should be contained in this report. For example, a user might want a report that includes all comments for FTLS files only; he must specify the FTLS as the scope for this report generation. Another example would be to ask for a report that includes all comments related to the Software Requirement Specification document. With the proper permission, a user also might be able to generate a report based on the information stored in other users' notecard files.

Future Directions

An example of some future advances in software technology that could be incorporated into this prototype is in the area of visual specification systems. An innovative perspective on this subject has been initiated by Dr. Jeanette Wing at Carnegie-Mellon University who is leading an effort called Miro' [7]. Miro' draws from David Harel's design abstraction known as "statecharts" [8], and centers on the MIT Larch Family of Languages and theorem prover. The goal of Miro' is to design a visual specification language applicable to security and concurrency.

With respect tc other verification environments, another notable reference is the Penelope Verification System of Odyssey Research Associates [9]. Many of the methods that are being developed and tested by this group are intended for code verification (a.k.a. program verification) and certainly would integrate well with the Specification Browser utility.

An enhanced design could incorporate a shared Configuration Management System (CMS). Modifications are certain to occur with any development effort, and how these decisions affect the various portions of the Verification Evidence could gracefully be handled by including CMS into this tool. By *integrating* the role of the software development CMS into the verification process, one could better manage the changes to the system. Synchronizing the updates to the shared CMS would require adding a system of communication between the two disciplines' view of the CMS interface and would enforce changes to remain in lock-step with one another.

Conclusion

The Specification Browser tool we propose would be easy to use and to learn. The learning curve could realistically be in terms of hours, thereby giving evaluators a genuine sense of accomplishment and dramatically impacting the pace at which individual contribution would take place.

We see the browser as a major factor in promoting a professional exchange between team members and vendors, while providing a less tedious medium for examining the verification technical matter. Additionally, the Specification Browser will increase the level of assurance in software systems. As technology advances, one-time burdensome tasks can turn into enjoyable work, and the mystique and stigma that surrounds the verification process may vanish. With the advent of well-developed verification environments, we could begin to see an increase in the pool of those who understand the proofs–thus opening up this discipline to many others. It is likely that this will have a positive effect on the field of computer security.

BIBLIOGRAPHY

1. Benzel, Terry C. V., "Analysis of a Kernel Verification," The MITRE Corporation, MTR 9213 (Volume 1), Bedford, MA., 1984.

2. "Department of Defense Standard: Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, Department of Defense, Washington, DC, December 1985.

3. "Military Standard: Defense System Software Development," DoD-STD-2167A, Department of Defense, Washington, DC, 4 June 1985.

4. Scheid, John, and Holtsberg, Steven, "The Ina Jo Specification Language Reference Manual," TM-6021/001/04, Unisys Corporation, September 1988.

5. Eggert, P., et al, "FDM User Guide," TM-8486/000/02, Unisys Corporation, 1988.

6. Knuth, Donald E., "Literate Programming," The Computer Journal, 27(2), 1984.

7. Wing, Jeanette M., "Visual Specifications of Software Systems," Carnegie-Mellon University, Pittsburgh, PA, 1988.

8. Harel, D., Pnueli, A., Schmidt, J. P., and Sherman, R., "On the Formal Semantics of Statecharts," Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, NY, pp. 54-64, 1987.

9. "Penelope - An Ada Verification System," Odyssey Research Associates, Inc., Press Release, Ithaca, NY, December 1988.

ENGINEERING RESULTS FROM THE A1 FORMAL VERIFICATION PROCESS

Timothy E. Levin
Steven J. Padilla
Roger R. Schell

Gemini Computers, Incorporated
P. O. Box 222417
Carmel, California 93922

ABSTRACT

The GEMSOS TCB, currently under development, is targeted for the class A1 level of the Trusted Computer System Evaluation Criteria. The formal methodology used to verify the security of the GEMSOS TCB is reviewed. Specific results from making the formal verification process an integral part of the engineering of the system are described. These results are shown to have significantly contributed to the security and integrity of the GEMSOS TCB. The concrete and definitive contributions of the formal verification reflected in the GEMSOS design choices are presented. These contributions are shown to provide more than just a vague sense of increased assurance.

OVERVIEW

The GEMSOS TCB under development is targeted for the TCSEC Class A1 level. As part of this effort, Gemini is formally verifying the TCB as specified in the Trusted Computer System Evaluation Criteria (TCSEC) [TCSC]. This verification includes the production of a formal security policy model (Model) and a formal top level specification (FTLS), the demonstration of correspondence between the FTLS and the TCB source code, and a covert storage channel analysis of the FTLS.

Since the business thrust of Gemini is on building commercial products [SHOCK1], major attention has been paid to the impact of the formal verification on the product. The experience has been that the formal work is much more that just an adjunct to provide evidence for an outside evaluation. Making the formal work an integral part of the engineering process has enhanced the quality as well as the security of the product.

Gemini has chosen the Unisys Corporation's Formal Development Methodology (FDM) system for verification and specification support. The GEMSOS Model and FTLS are written in the FDM Ina Jo specification language [SCHEID]. The FDM Interactive Theorem Prover (ITP) [SCHORR] is used to prove (1) the Basic Security Theorem (of Bell and LaPadula) with respect to the policy defined in the constraints and invariants of the model and (2) that the FTLS is consistent with the Model. The Ina Jo specification of the FTLS is the basis for both the code correspondence demonstration and the covert storage channel analysis.

The GEMSOS TCB is partitioned into a kernel layer that implements a reference monitor for the mandatory access control policy [SCHELL] and a non-kernel layer that enforces the discretionary access control policy. Each of these layers enforces a separate security policy and each is verified through a separate Model and FTLS; the policy enforced at the TCB interface is a combination of the policy subsets enforced by the two layers [SHOCK2].

Verification of the TCB has taken place concurrently with development of the TCB. In this strategy, the Model and FTLS are written primarily while the interface design (B-spec) for the layer is written. Preliminary Model proofs, FTLS proofs, and covert channel analysis are performed during development (C-spec and coding), while code correspondence, covert channel measurements and final proofs are perfo med after completion of TCB code.

As this paper is being written, the formal verification of a pre-evaluation version of the GEMSOS kernel has been completed in all phases. The specifications of the Model and FTLS for the non-kernel TCB are in progress.

The interaction of the engineering and formal verification efforts has been encouraging. We have seen positive feedback involving all areas of the formal process through all phases of the development process. This feedback has been effective in both directions:

1) it has allowed the verification work to remain accurate while progressing concurrently with engineering,

2) it has provided formulative and corrective guidance to the TCB design and implementation.

It is this second direction of input that is the focus of this paper, i.e., how the formal verification process has contributed to the security and correctness of the TCB design and implementation.

FORMAL METHODOLOGY DESCRIPTION

The goal of TCB verification is to provide assurance that a TCB implements a stated security policy. To attain this goal, a chain of formal and informal evidence is produced which is composed of statements of TCB functionality, each at a different level of abstraction, along with assertions that each statement is valid with respect to the next most abstract statement. The sequence of functional statements are the security policy (Policy), the formal security policy model (Model), the top level specification (Specification), the TCB source code (Source), and the TCB itself (binary and hardware). The result of the chain of evidence is an overall transitive assertion that the TCB implements the Policy. This chain of evidence is illustrated in Figure 1.

The Model is the linchpin of the argument. It is not merely a formal statement of the Policy, nor just a precise mathematical statement of the security functions of the TCB. Its critical characteristic is that it is a model of a reference monitor. This implies that by demonstrating just the TCB is a valid interpretation of the Model, it is shown that the entire computer system is secure. In particular, this implies that the (untrusted) hardware and software that is outside the TCB, and thus not modeled, cannot result in access to information in violation of the Policy, since the chain of evidence was produced without dependence on these untrusted components.

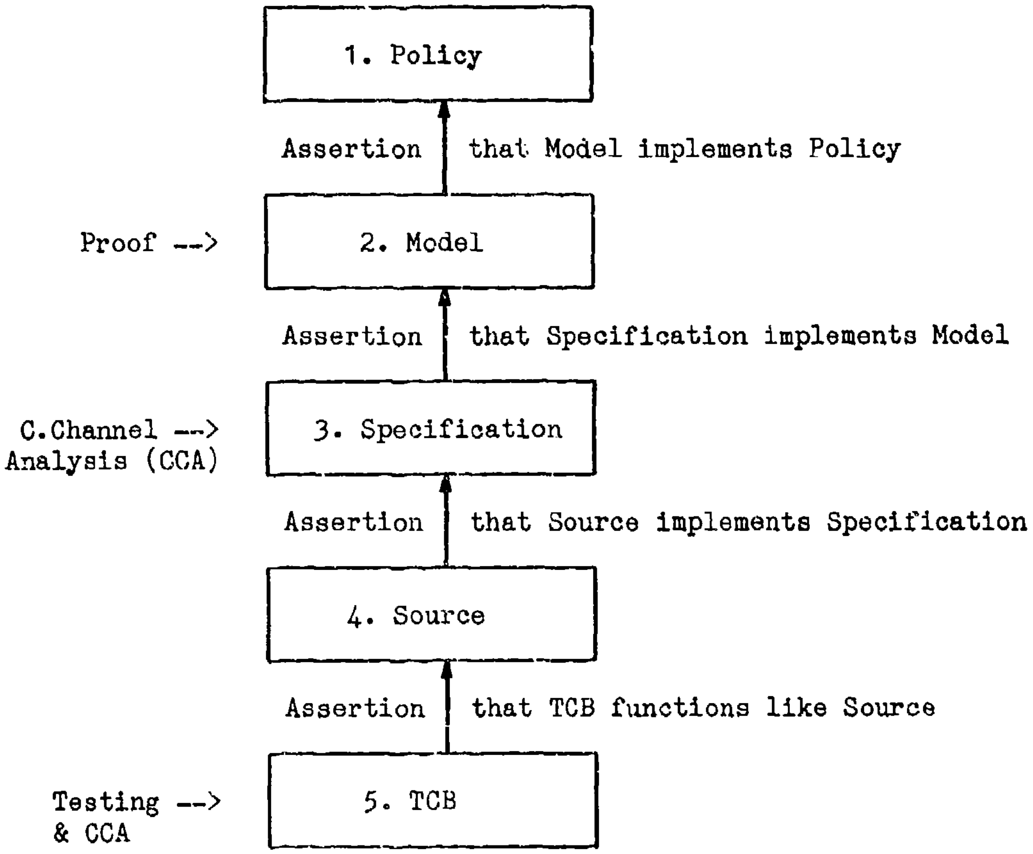

1. Policy
Assertion that Model implements Policy
Proof -->
2. Model
Assertion that Specification implements Model
C.Channel -->
Analysis (CCA)
3. Specification
Assertion that Source implements Specification
4. Source
Assertion that TCB functions like Source
Testing -->
& CCA
5. TCB

Figure 1. Chain of Verification Evidence

TCSEC Requirements

For the verification of a Class A1 system, the TCSEC requires (1), the above-mentioned chain of evidence including a "formal" top level specification (FTLS), (2), a set of empirical validations of the TCB functional statements, and (3), a set of descriptive specifications.

The requirements for the chain of evidence consist of functional statements one through five (from figure 1) along with connecting validation assertions prepared using specific techniques. The requirement for the Model-to-Policy assertion is satisfied by an informal discussion. The requirement for the Specification-to-Model assertion is satisfied by a correspondence demonstration using a combination of informal and formal techniques. The requirement for the Source-to-Specification assertion is satisfied by an informal (code-correspondence) demonstration. The TCB-to-source assertion (i.e., compiler and hardware validation) is considered beyond the "state of the art" and is not required by the TCSEC (except indirectly via testing).

The required empirical statement validations (see Figure 1) are a proof that the Model is consistent with its security assumptions (axioms), a covert channel analysis of the FTLS and the TCB, a direct testing of the TCB interface, and a testing of the TCB covert channels.

The descriptive (informal) requirements of Class A1 verification consist of a descriptive top level specification (DTLS), descriptions of various aspects of TCB security, and descriptions of how the DTLS relates to the Model and the TCB.

In this discussion, the chain of evidence between the Model and the TCB source code, and the corresponding empirical validations, are considered to be the formal verification of the TCB.

GEMSOS Formal Verification Components

The GEMSOS Model and FTLS are specified in the FDM Ina Jo language. The FDM system allows specifications to be related in "levels." Ina Jo includes a facility for formally mapping the elements of one level to the elements of the next level. The specification of a given level can then be shown to support the properties (e.g., security) of the level above it. The uppermost level is used to state the security criteria for the system. Lower layers are usually written at a less abstract level than the upper layers and are used to provide concrete functional detail about the TCB interface.

Model and Proof The GEMSOS Formal Security Policy Model is written as the topmost Ina Jo level, with the FTLS the next level. The Model is a mathematical statement of the GEMSOS access control policy. The MAC portion of the model [LEVIN1] states the mandatory access control policy and the DAC portion [LEVIN2] states the discretionary access control policy.

The Model is "proven" to uphold the policy using the FDM Interactive Theorem Prover (ITP). The Ina Jo processor produces theorems based on input specifications. The theorems are used as inputs to the ITP. The theorems state that the model rules (transforms) preserve the security conditions defined in the model. The ITP negates each theorem, providing groundwork for a proof by contradiction. The proof of the Model thus shows that the Model rules uphold the TCB security policy, viz., that Model objects are only accessed according to policy. This is done for both the MAC and DAC portions of the Model.

The Ina Jo constants, variables and criterion of the Model define the GEMSOS interpretation of the Bell and LaPadula security model [BLP]. Ina Jo transforms are used to express the Model "rules."

FTLS and Proof The GEMSOS FTLS for each of the two TCB subsets is written to reflect exceptions, error messages and effects visible at the interface(s) of the GEMSOS TCB. The GEMSOS kernel FTLS specifies the kernel interface (reflecting the MAC Policy) and the GEMSOS TCB FTLS specifies the TCB interface (reflecting the DAC Policy). The ITP is used to prove that the FTLS upholds the security properties of the Model level.

At the FTLS level, there is a transform corresponding to each call plus two hardware transforms which abstractly represent the hardware "read" and "write" operations. Each of these TCB calls is shown to map to a rule of the model. Each FTLS level transform is structured as a large conditional followed by a no-change statement. Within each conditional, exceptions are specified in the order of occurrence in the corresponding call; the last case of the conditional contains the primary change statements of the transform. The ordering of exceptions is significant during covert storage channel analysis.

Code Correspondence The code correspondence between the FTLS and the TCB source code shows that the code is a valid interpretation of the FTLS, and therefore upholds the security properties of the FTLS. The GEMSOS code correspondence report consists of three parts:

1. A description of the correspondence methodology

2. An accounting of the non-correlated source code

3. A map between the elements of the FTLS and the TCB code

The TCSEC requirements suggest two aspects of correspondence: the FTLS accurately describes (viz., corresponds to) the TCB and the TCB is consistent with (viz., a valid interpretation of) the more abstract FTLS. A specific objective of the consistency requirement is to ensure that all security relevant functions of the TCB are represented in the FTLS. Thus, any deliberate or accidental "trap door" in the code is detected and identified.

Covert Storage Channel Analysis Covert storage channel analysis of the FTLS shows that non-data information flows do not violate the security policy. All information flow into and out of objects mapped to the Model (i.e., data flow) is accounted for in the FTLS. In the proof of the FTLS data flow is shown to conform to the security policy. Non-data information flows typically involve attributes of the system and of objects rather than data itself. These are typically transmitted outside of the TCB through returned error codes.

The analysis[LEVIN3] utilizes the shared resource matrix (SRM) methodology of Kemmerer[KEMMER].

ENGINEERING RESULTS

Feedback to the development process has occurred in both the kernel and non-kernel areas of the TCB. Below, several specific cases in each area showing how this feedback has been beneficial are discussed.

TCB Kernel

In the kernel, primary input to the development process occurred as the result of code correspondence, Model and FTLS proofs, and covert channel analysis.

Code correspondence Code correspondence of the kernel revealed two significant instances where the implementation didn't match the specification.

In the process_create call, a series of segments are passed to the child process as parameters. Other parameters define the modes and privilege levels of the segments which will be "made known" (i.e., made accessible) in the child's address space when it starts execution outside of the kernel. In the implementation, the segments were being made known for the child with the privilege level that the parent had, rather than the privilege specified in the parameters. This problem was discovered through code correspondence of the segment manager layer of the kernel. It can be assumed that the error may also have been caught during the functional testing of the kernel. The problem was subsequently fixed through implementation of an Engineering Change Proposal.

The second discrepancy involved the implementation of current access. The Bell and LaPadula [BLP] notion of current access to an object is defined in the FTLS as a subject having access to a valid segment selector for the object in the local descriptor table (LDT). The makeknown_segment kernel call is designed to put a segment into a subject's current access set. However, during code correspondence it was discovered that the LDT was not actually set during makeknown, but rather was set during the swapin_segment kernel call that loads the segment into RAM. This difference was known to the engineering staff, but had been overlooked as changes were made to the kernel. It was subsequently fixed through implementation of an Engineering Change Proposal.

Proofs The proofs of the Model and FTLS are designed specifically to demonstrate that the security policy of the TCB is upheld by the TCB operations. During the proofs, several overt security flaws (as opposed to covert channels) were discovered in the kernel.

In the GEMSOS TCB, a subject is a process, ring pair. Each process is made up, then, of a subject in each of eight rings. In addition to activating or deactivating subjects, the activate_subject kernel call can be used to change the label range for one or more subjects of a process. (Each subject has a "read_class" and a "write_class" label [DENN1N] that form a range where read_class always dominates write_class. For untrusted subjects, that range is nil.) The label range of each subject of a process must enclose the range of all of the subjects of less privilege in that process (see Figure 2). In the event that this enclosure is not correct as the result of a change to a subject's range, the kernel MOVES the less privileged subject's labels so that they are properly enclosed.

```
Ring 3 Subject Range            READ WRITE
Ring 2 Subject Range        READ          WRITE
Ring 1 Subject Range    READ                   WRITE

(labels to the left dominate labels to the right)
(subject n is more privileged than subject n+1)
```

Figure 2. Subjects of a Process with Enclosing Label Ranges

The label range enclosure is required due to the use of hardware descriptors to enforce current access and the implementation of those descriptors in the iAPX286. In the iAPX286 a task (which maps to an active subject at the kernel interface) is allowed the use of any descriptor which has a privilege level at or above its own privilege level (e.g., a task in privilege level 1 can use a descriptor in privilege level 3 to access a segment). Thus, the Ring 1 Subject in Figure 2 can access all the segments that are accessible to the Ring 2 subject in Figure 2. If the Ring 2 subject had a range which was greater than (or incomparable with) the range of the Ring 1 subject, then the labels of the Ring 1 subject might not permit access (from a security policy point of view) to some object that the Ring 2 subject could access. If the Ring 2 subject were to gain access to such an object then (due to the descriptor mechanism) the Ring 1 subject would also gain access to that object and a violation of policy would occur.

During the proof of the FTLS, it was discovered that an outer ring subject could keep a label range that was not enclosed by the new range of the more privileged subjects beneath it. This was due to inverted logic in a dominance check that didn't MOVE the outer ring labels in the case where they were incomparable with the new inner ring labels. The InaJo specification of the incorrect design is:

```
dominates (ring_3_read_class, new_ring_2_read_class)
        then move (ring_3_read_class)
and
dominates (new_ring_2_write_class, ring_3_write_class)
        then move (ring_3_write_class)
```

This problem was subsequently fixed through an Engineering Change Proposal to implement the following InaJo specification:

```
~dominates (new_ring_2_read_class, ring_3_read_class)
        then move (ring_3_read_class)
and
~dominates (ring_3_write_class, new_ring_2_write_class)
        then move (ring_3_write_class)
```

In another iteration through the proof of the activate_subject call, it was discovered that the updated "MOVE" operator did not ensure that the outer ring subject's new label range was correct, (i.e., the new read class did not dominate the new write class) even though the inner ring subject's new label range was correct. This problem was subsequently fixed through implementation of an Engineering Change Proposal to the MOVE operator.

Covert Channel Covert storage channel analysis of the GEMSOS kernel revealed two unexpected information flows. Both problems occurred in the dismount_volume call. Dismount_volume is used to temporarily remove a set of segments from the segment structure. When using GEMSOS, segments can only be made known from a mounted volume.

The first problem occurred because the dismount_volume call returns an error if any segments on the volume are madeknown by any subject. Although the security checks (made during dismount) ensured that the calling subject could both read and modify the segments on the volume, the checks did not ensure that the calling subject had the proper label range to read the LDTs for the segments in all processes. (It was possible for a volume which contained only unclassified segments to be dismounted by an unclassified subject. However, a top secret subject could makeknown one of the segments on the volume and thus cause the dismount call to fail due to the fact that the top secret subject had a valid LDT for a segment on the unclassified volume.) This problem was subsequently fixed through implementation of an Engineering Change Proposal that required, for dismount, that the calling subject's labels (i.e., read class and write class) must range from volume low to system high.

The second covert storage channel found in dismount volume was less dramatic and involved the order of exception checking. Several conditions about the volume were checked before the calling subject's labels were compared to the volume labels. This meant that errors about the volume could be reported to the calling subject, even though that subject might not have the proper authority to view that information. This problem is being fixed through implementation of an Engineering Change Proposal that requires the calling subject's label range to be checked before returning errors relative to the volume condition.

Non-Kernel TCB

In the verification of the non-kernel TCB, primary input to the development process has occurred while in the conceptual phase of writing the Model and FTLS, and in providing a formal mapping between the Model and the FTLS. A [illegible]jor benefit has been that the designers are required to use clean abstractions that can be readily represented in the FTLS. This has forced the developers to avoid designs that would be difficult to evaluate (and understand). It is difficult to identify the various poor designs that were avoided, but the following examples of design analysis will illustrate the value of the formal methods to the design process.

One example can be seen in the process create call to the TCB. This call creates a child process and provides the child current access (i.e., that relationship represented by the Bell and LaPadula "b" set) to a set of objects. In producing the Model and FTLS of the process_create call, it was realized that the permission (i.e., that relationship represented by the Bell and LaPadula "M" matrix) to the objects passed to the child was not being checked. The problem is, while the parent must have current access to the objects passed, this does not guarantee that it has permission to them at the time that it passes them to the child. (In GEMSOS, permission is asserted each time current access is gained, but not during access.) Thus, current access was being propagated across process boundaries, where permissions for the child process had possibly been revoked. This problem was fixed through a change to the functional specification (i.e., B-spec) such that the TCB is now required to check the child process's permissions to the objects madeknown for it.

The work on the FTLS resulted in a change to the design of the ACL checking function. The [illegible] level design was described in terms of a bit manipulation algorithm (i.e., [illegible] and XORs). In trying to describe this design formally, it was decided that [illegible]earer, more understandable design should be pursued. As a result, an alt[illegible]sign utilizing PASCAL sets was adopted that was not only much eas[illegible]rstand and specify, but was also much simpler and more efficient to impl[illegible].

Another problem was discovered in mapping the FTLS to the Model. It was realized that the named objects of the DAC Model were not being uniformly treated as objects at the TCB interface. Named objects are those objects to which DAC is applied. The named objects of the TCB are discretionary access control nodes (dacls) and multi-segments (msegs). Segments are not named objects at the TCB interface, although they make up portions of msegs and can be made known individually. The TCB interface calls did not provide an operation to create or makeknown msegs. Rather, this was done by implication as the result of creating or making-known the mseg root segment (msegs are made up of trees of segments). It was decided that the interface would present a cleaner abstraction of objects if there were explicit calls to create and makeknown the objects (i.e., msegs). The create_mseg and makeknown_mseg calls were added to the DTLS.

The Model and FTLS also allowed us to focus on the policy (DAC) supported by the non-kernel TCB. For example, the modes of access granted to any given segment (a part of an mseg) is of no interest to the TCB after the initial check is made to see that the modes are a subset of those granted when the mseg was made known. This per-segment information was originally being maintained in the TCB; however, once the checks are complete and a descriptor is created for the segment, subsequent access to the segment is controlled directly by the hardware. The kernel, on the other hand, maintains this information and uses it in subsequent calls such as mount, dismount, and makeknown where the individual segment will be used in naming other segments. Since the DAC policy does not require this information (i.e., the segment access modes) to be maintained, it was removed from the DTLS, simplifying both the specification and the implementation.

CONCLUSION

Formal verification of the GEMSOS TCB has helped significantly in discovering conceptual and implementation errors that may have otherwise been overlooked or carried forward. Formal verification has included the production of the formal security policy model and the formal top level specification, specification-to-model mapping, code correspondence of the FTLS and covert storage channel analysis of the FTLS.

An important contribution of the formal verification is the high confidence that non-secure information flows will be detected in the design. Errors in the initial TCB implementation have been discovered in all phases of formal verification. These discoveries and their subsequent corrections have been instrumental in ensuring the security and integrity of the GEMSOS TCB.

Of perhaps even greater importance is the somewhat subtle but pervasive impact on the designers of requiring a design that can be easily specified in an FTLS that must be mapped to the implementation. By making formal verification an integral part of the engineering process, the set of design alternatives that naturally emerge are those that are easily evaluated. The practical experience in a commercial product development strongly supports the conclusion that there should be significantly higher confidence in the security of a system developed with formal methods (viz., Class A1) than a comparable system (viz., Class B3) developed without them.

REFERENCES

[BLP] Bell, D.E. and LaPadula, L.J. , "Computer Security Model: Unified Exposition and Multics Interpretation," Tech. report ESC-TR-75-306, MTR-2997 Rev.1, The Mitre Corporation, Bedford, Mass., March 1976

[DENNIN] Denning, D.E., Lunt, T.F., Schell, R.R., Heckman, M., and Shockley, W.R., Secure Distributed Data Views (SeaView): The SeaView Formal Security Policy Model. Computer Science Laboratory, SRI International, July, 1987.

[KEMMER] Richard A. Kemmerer, "A Practical Approach to Identifying Storage and Timing Channels," IEEE Report CH1753-3/82/0000/066, 1982.

[LEVIN1] Levin, T., Padilla, S., "Formal Security Policy Model for the GEMSOS Kernel," February 1989, Gemini Computers, Inc., Technical Report GSC-89-03-01.

[LEVIN2] Levin, T., Padills, S. and Irvine, C., "A Formal Model for UNIX Setuid" in Proceedings of the 1989 IEEE Symposium on Security and Privacy, May 3-5, 1989, Oakland CA., pp. 73-83.

[LEVIN3] Levin, T., Padilla, S., "Covert Storage Channel Analysis of the GEMSOS Kernel," March 1988, Gemini Computers, Inc., Technical Report GCI-88-09-01.

[SCHEID] Scheid,J., Anderson, S., Martin, R., and Holtzberg, S., "The Ina Jo Specification Language Reference Manual—Release 1." TM 6021/001/02. System Development Corporation, Santa Monica, Ca., 1986.

[SCHELL] Schell, R.R., Tao, T.F., and Heckman., M, "Designing the GEMSOS Security Kernel for Security and Performance", in Proc. Eighth National Computer Security Conference, Gaithersberg, MD, October 1985, pp. 108-119.

[SCHORR] Schorre,V. and Stein, J., "The Interactive Theorem Prover User Manual," TM 6889/000/05, System Development Corporation, Santa Monica, Ca., September 1984.

[SHOCK1] Shockley, W.R., Tao, T.F., and Thompson, M.F., "An Overview of the GEMSOS Class A1 Technology and Application Experience", in Proc. 11th National Computer Security Conference, 17-20 October, 1988, Baltimore, MD, pp. 238-245.

[SHOCK2] Shockley, W.R. and Schell, R.R., "TCB Subsets for Incremental Evaluation", in Proc. 3rd Aerospace Computer Security Conference, 1987, American Institute of Aeronautics and Astronautics, Washington, D.C.

[TCSC] Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985.

GUIDELINES FOR FORMAL VERIFICATION SYSTEMS: OVERVIEW AND RATIONALE

Monica McGill Lu
National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755-6000

Barbara A. Mayer
Trusted Information Systems, Inc.
3060 Washington Road (Rt. 97)
Glenwood, MD 21738

ABSTRACT

The *Guidelines for Formal Verification Systems* documents the procedures for NCSC endorsement of verification systems. This paper describes the history and current status of the Guidelines, the endorsement process, the evaluation approach, the major qualifications of and the possible future directions for verification systems. The purpose of this paper is to inform verification tool and trusted system developers of the current endorsement process, the rationale behind it, and how it may affect the verification community.

INTRODUCTION

The National Computer Security Center (NCSC) recently published a guideline to be used in evaluating formal verification systems for possible placement onto the Endorsed Tools List (ETL). This guideline, *Guidelines for Formal Verification Systems* (the Guideline), is the culmination of several levels of effort.

This paper focuses on the rationale behind the Guideline and how it could affect the developers of formal verification systems who are interested in having their systems evaluated for endorsement. A history of the development of the Guideline is given, followed by an overview. The overview describes each of its major sections, including the evaluation process, the major qualifications to be evaluated, and possibie future qualifications for verification systems.

BACKGROUND

The *Trusted Computer System Evaluation Criteria* (TCSEC), DoD 5200.28-STD [1], and the *Trusted Network Interpretation of the TCSEC* (TNI) [3] are the criteria used for evaluating security controls built into Automated Information Systems and network systems, respectively. The TCSEC and TNI classify levels of trust for computer and network systems by defining divisions and classes within divisions. Currently, the class providing the most trust, A1, requires formal design specification and verification. As stated in the Design Specification and Verification requirement in the TCSEC and TNI, "...verification evidence shall be consistent with that provided within the state-of-the-art of the particular Computer Security Center-endorsed formal specification and verification system used."

The earliest notion of what it meant for a verification system to be "NCSC-endorsed" was rather loose. The authors of the TCSEC wanted to emphasize that evaluators of trusted systems are responsible for evaluating verification evidence, but are not responsible for evaluating the basis for the evidence. Evaluators are not responsible for having to learn and understand novel, unfamiliar, or untried verification systems in order to evaluate the verification evidence.

The authors also wanted to restrain the proliferation of clones of existing verification systems. An "NCSC-endorsed" verification system is to be unprecedented and innovative. The Computer Security Center Product Evaluation Program documentation states,

> "[a]n Endorsed Tools List [ETL] is maintained by the CSC [Computer Security Center]. The ETL specifies the tools and versions that are currently supported. The current set of CSC-endorsed verification tools may expand or contract as the need arises. A compelling reason wo ld have to exist to justify the addition of a verification tool to the endorsed tools list -- the proposed tool would have to offer some significant feature not provided by the current set of endorsed tools." [2]

The endorsement process was to be conducted via a social process. A verification system would be endorsed as a result of usage, supportability, and acceptance by the verification community. At that time, the endorsed verification systems included the Formal Development Methodology (FDM), the Gypsy Verification Environment (GVE), and the Hierarchical Development Methodology (HDM). It eventually became clear that, in order to implement an effective program, a more rigorous definition of the endorsement process was needed.

The NCSC established a Verification Committee in June, 1986, to enact policy decisions regarding the endorsed tools. The Committee consists of the NCSC Deputy Director (who serves as its chairperson), NCSC Chief Scientist, representatives from the NCSC's Office of Research and Development and Office of Computer Security Evaluations, Publications, and

Support. The Committee is responsible for adding and deleting tools from the ETL, as well as making policy decisions regarding the NCSC's verification research and support programs.

In conjunction with the 1987 IEEE Symposium on Security and Privacy, NCSC representatives held a Birds of a Feather session to present the first attempt at "factors" for endorsement. Rather than discussing the factors, the meeting focused on the existence of an ETL. A few individuals opposed the notion of an ETL and expressed the concern that rating or ranking verification systems, which are largely research tools lacking production-quality features, would not be possible.

In concert with these efforts, the Committee directed the Publications Division of the NCSC in 1988 to complete and publish the *Endorsed Tools List* and the *Guidelines for Formal Verification Systems*. The first publication of the ETL was in January 1989.[4] The Guideline underwent three extensive reviews. The final draft was reviewed by over forty individuals in the verification community. After incorporation of the comments on the final draft, the Guideline was approved for publication by the NCSC Director in April 1989.[5]

OVERVIEW OF THE GUIDELINE

The Guideline has five major sections. The first describes the evaluation and endorsement process. The second and third define the technical and support requirements, respectively. The fourth contains a list of possible future directions, while the final section consists of a glossary of terms.

The Guideline defines requirements that can and should exist in current verification technology for production-quality systems. A production-quality verification system is defined as one that is sound, user friendly, efficient, robust, well-documented, maintainable, developed with good software engineering techniques, and available on a variety of hardware.[5] The Guideline addresses only verification systems that provide automated support, although there are manual methodologies for performing formal specification and verification.

Evaluation Approach

The verification system developers play a crucial role in the evaluation process. Developers need to be available to answer questions, provide training, and meet with the evaluation team to discuss outstanding issues. Beyond support, their degree of participation depends on which one of the types of evaluations is being performed.

The types of evaluation processes are:

- evaluation of new verification systems being considered for addition onto the ETL,
- evaluation of new versions of systems already on the ETL for addition onto the ETL (reevaluation for endorsement), and
- reevaluation of systems on the ETL being considered for removal from the ETL (reevaluation for removal).

One of two types of reports is issued at the end of the evaluation. The type of report issued depends on the type of evaluation performed. Each report fully documents the evaluation team's findings. Upon completion of the evaluation, these documents are available to both the developers and the users.

The ETL is updated when a new system or version of a system is added or a system is removed. If a new version of a verification system that already exists on the ETL is endorsed, the new version is added to the ETL and the old version is archived as a previously endorsed version.

__Evaluation of a New System__: A new system is considered for evaluation if it provides some significant feature or improvement that is not available in any of the currently endorsed tools. Upon determination that this condition is met, the evaluation team (assigned by the Verification Committee Chairperson) analyzes the verification system, concentrating on the qualifications sections (sections 3 and 4) of the Guideline. Studies or prior evaluations performed on the verification system, as well as any history of use, shall also be considered when evaluating the verification system.

Upon completion of the evaluation, a TAR is written by the evaluation team addressing each of the qualifications discussed in the Guideline. The TAR is presented to the Verification Committee, and the Committee Chairperson makes the final decision of endorsement based on the Committee's recommendation. If the system is endorsed, the ETL is revised and issued to include the newly endorsed system.

__Reevaluation for Endorsement__: A reevaluation for endorsement may be warranted after significant amounts of change or after enhancements to a currently endorsed verification system have been made. The intent of this type of reevaluation is to permit improvements to endorsed versions and advocate state-of-the-art technology on the ETL while maintaining assurance of the original endorsed version. The developer is responsible for submitting evidence that the improvements to the system have not affected the soundness or integrity of the system. This evidence is summarized in the form of a VR. The VR assures that only listed changes have been made and unchanged code is not affected by the changes. Additionally, the VR includes sufficient commentary to allow an understanding of every change made to the verification system as well as the implications of the changes.

The evaluation team is responsible for the final evaluation of the system. The evaluation team's primary responsibilities are to review the VR thoroughly and test the functionality of the changes. Upon completion of their analysis and discussion of their findings with the Committee, the Committee Chairperson approves or disapproves endorsement based on their recommendation.

__Reevaluation for Removal__: Once a verification system is endorsed, it shall normally remain on the ETL as long as it is supported and is not replaced by another system.

Reasons which may warrant removal of a verification syste from the ETL are too many bugs, no users, elimination of support and maintenance, and unsoundness. The verification community (including the Committee) may question the endorsement of a verification system on the ETL. Upon bringing this to the attention of the Committee, an evaluation team begins a reevaluation of the system, focusing on the area in question.

Upon completion of the reevaluation for removal, . TAR is written by the evaluation team addressing each of the concerns that instigated the reevaluation for removal. The TAR is presented to the Verification Committee, and the Committee Chairperson makes the final decision on removal based on the Committee's recommendation. If the system is to be removed, the ETL is revised and issued to exclude the removed system.

Beta Versions: The version of the verification system that has been endorsed may not be the newest and most capable version. These intermediate versions are not endorsed and are known as "beta" tool versions. The goal of beta versions is to stabilize the verification system before its submission for evaluation. Beta versions are useful in helping system developers uncover bugs before submitting the verification system for evaluation.

Users should not assume that any particular beta version will be evaluated or endorsed by the NCSC. If the developer of a trusted system is using a beta version of the formal verification system, specifications and proof evidence shall be submitted to the NCSC which can be completely checked without significant modification using an endorsed tool as stated in the A1 requirement. This can be accomplished by using either the currently endorsed version of a verification system or a previously endorsed version that was agreed upon by the trusted system developer and their evaluation team. Submitted specifications and proof evidence which are not compatible with the endorsed or agreed upon version of the tool may require substantial modification by the trusted system developer.

Main Categories of Factors

The technical factors for endorsed verification systems are divided into two major categories:

- methodology and system specification, and
- implementation and other support factors.

The methodology and system specification section covers the underlying principles as well as specific features, assurances, and documentation requirements. Other support factors include user interface, hardware and maintenance support, configuration management, testing, and docu mentation requirements for the implementation factors. The Guideline was divided into these two main categories to address these separate issues.

Methodology and System Specification: The Guideline divides verification systems into a minimal set of components that are necessary to perform design verification. These components were chosen since they are found in current verification systems and include:

- a mathematical specification language that allows the user to express correctness conditions,
- a specification processor that interprets the specification and generates conjectures interpretable by the reasoning mechanism, and
- a reasoning mechanism that interprets the conjectures generated by the processor and checks the proof or proves that the correctness conditions are satisfied.

The methodology or underlying principles and rules of organization of the verification system provide a sound, logical foundation for the verification system. For this reason, the requirement for a methodology is a necessary but not sufficient condition for endorsement.

The features, assurances, and documentation requirements extend across each of the components of the verification system. The requirements dictate that each component provide sufficient functionality and assurance, both technically and in the form of documentation, to demonstrate that it works correctly and collectively. These factors were chosen because the state-of-the-art verification systems have the capability to implement each of these factors.

For example, the specification language should be sufficiently expressive to support the methodology of the verification system. It also should include precisely defined and documented syntax and semantics. Correctness conditions need to be expressible.

The specification processing component should be able to accept as input the constructs of the specification language and should be able to convert the specification into a form or language that is acceptable to the reasoning mechanism. Conjectures derived from the correctness conditions should also be generated.

Current verification systems lag behind the state-of-the-art in theorem proving. The factors for the reasoning mechanisms were specifically chosen to elevate their functionality to meet the state-of-the-art. The reasoning mechanism should be capable of processing the conjectures produced by other components. Additionally, it should provide a means of documenting, reprocessing, reusing, and validating proofs.

Implementation and Other Support Factors: Support factors are measures of usefulness, understandability, and maintainability of the verification system. For example, the user interface should be user-friendly, providing understandable input and informative output. The hardware that the verification system runs on should not be obscure or obsolete and the processing efficiency should be reasonable.

In order for a verification system to be adequately maintained, ongoing support should be available. Additionally, comprehensive testing should be performed and supporting documentation should be available

The most extensive factor in this category is configuration management. A configuration management plan, along with the supporting procedures and tools, is essential to demonstrate that additions, deletions, or changes made to the verification system do not jeopardize its soundness or its ability to satisfy any of the requirements. Configuration management also ensures that changes made to the verification system takes place in an identifiable and controlled environment. The core of this requirement was derived from *A Guide to Understanding Configuration Management in Trusted Systems*.[6]

Future Directions and Beyond A1

During the review process of the draft Guideline, a few individuals remarked that the future directions section was not appropriate for the Guideline and should be removed. This section was added and kept as part of the Guideline for several reasons. First, the section demonstrates that the NCSC is looking at verification systems for levels of assurance "beyond A1". Second, many of the reviewers commented on the lack of concern with code verification in the Guideline. The primary goal of the document is for design verification, so code verification is addressed in the future directions section. Third, the NCSC encourages the research and development of new verification systems, whether or not they are targeted for beyond A1.

This section is not intended to limit areas of future research. The list merely contains possibilities for future research -- areas which researchers *may choose* to investigate, such as code verification, hardware verification, high-level debugging, and concurrency. The NCSC recognizes that there are many other directions for verification research that are not mentioned on the list, and strongly encourages these as well.

Glossary of Terms

Although small in size, the glossary required a considerable amount of concentrated effort. The glossary covers terms used throughout the document in an attempt to assure that the readers have a common basis for understanding the Guideline. Certain definitions had to be incorporated, since those definitions appear in the TCSEC (i.e., formal verification, verification). Other terms were derived from standard textbook definitions. For those not appearing in the TCSEC or standard textbooks, definitions had to be created and repeatedly fine-tuned.

CONCLUSION

The extensive history involved in the development of the Guideline indicates the amount of effort that was expended by the NCSC as well as the verification community. From the numerous reviews, the breakdown of the threε types of evaluations evolved. After a determination of the type of evaluation to take place, the evaluation proceeds with emphasis on the major qualifications discussed in the Guideline. In addition, the future directions section discusses many possibilities that the developers may or may not incorporate into verification systems.

The Guideline will be updated to keep it consistent with the state-of-the-art of verification systems; therefore, the NCSC encourages feedback from the verification community. The NCSC's efforts in and support of current as well as future verification systems is ongoing.

REFERENCES

[1] National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[2] Department of Defense Computer Security Center, *Product Evaluation Program*, August,1984.

[3] National Computer Security Center, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, NCSC-TG-005, 31 July 1987.

[4] National Security Agency, *Information Systems Security Products and Services Catalogue*, Issued Quarterly, January 1989.

[5] National Computer Security Center, *Guidelines for Formal Verification Systems*, NCSC-TG-014-89, 1 April 1989.

[6] National Computer Security Center, *A Guide to Understanding Configuration Management in Trusted Systems*, NCSC-TG-006, March 1988.

COMPARING SPECIFICATION PARADIGMS: GYPSY AND Z

William D. Young
Computational Logic, Inc.
1717 W. 6th Street, Suite 290
Austin, Texas 78703

The application of formal methods to the analysis of computing systems promises to provide higher and higher levels of assurance as the sophistication of our tools and techniques increases. But evolution of the state of the art of formal program analysis is matched by increasing demands upon the technology. In the security arena advances in program verification methodologies, automated reasoning systems, specification techniques, and security modeling have been met with continuing reassessment of acceptable levels of assurance. System developers contemplating certification at the A1 level as outlined in the *Trusted Computer Systems Evaluation Criteria* [3], for example, can expect that the assurance requirements will become more rigorous with each year that passes.

Conversely, the desire for enhanced assurance drives the evolution of tools and techniques for providing it. One way to assure that technology keeps pace with evolving expectations is by continually re-evaluating our entrenched tools and techniques in relation to possible alternatives. The alternation of evaluation with informed refinement and selection can incrementally improve the research environment for all.[1] The verification community has been quite willing to compare and contrast various technologies and systems [15, 2, 16, 13] though it is unclear how much these comparisons have led to specific changes.

One of the most entrenched tools for providing assurance in the security area is the Gypsy Verification Environment [7] (GVE). The GVE is one of two systems endorsed by the National Computer Security Center for use in meeting the verification requirements for A1 certification. It has been used extensively in secure system specification and verification projects including the Encrypted Packet Interface [21], Message Flow Modulator [8], Honeywell SCOMP [5], Honeywell LOCK [1], and ACCAT Guard [14].

The Z specification language [9, 22] evolved within the Programming Research Group at Oxford University. We are not aware of its use within the security community though it has been used to specify significant software systems including a subset of the Unix filing system [18], the Computer Aided Visitor Information and Retrieval System [4], the ICL Data Dictionary [23], and a CICS System at IBM in the U.K. These examples suggest that Z might provide a viable specification language for secure systems. One goal of our research was to investigate this suggestion.

We present a comparison of the Gypsy and Z specification languages in the context of a nontrivial example. Our example is a previous specification of a subset of the Unix file system functionality [18][2] in Z and the translation of this specification into Gypsy. We compare and contrast the two specifications. On the basis of this comparison, certain conclusions are drawn which we hope can suggest refinements to the two languages and possibly a direction for future language designs which will avoid the pitfalls and capitalize on the strong points of each.

[1] This paradigm of scientific process is often blocked by prejudice for or against certain research directions, the personal and financial investments researchers and user communities have in those directions, and the momentum of ongoing system development projects.

[2] We are experts in the use of Gypsy but did not feel confident to write a creditable Z specification. We chose a problem which had been specified by Z experts to present Z favorably; we wish to make it absolutely clear from the outset that all of the Z text in this paper is taken almost verbatim from [18]. It is included here only to make the current presentation self-contained.

The Two Languages

Gypsy

Gypsy [6][3] is a *program description language* composed of two strongly intersecting components: a programming language and a specification language. Some parts of the language are used for programming, some for specification, and some for both. Among other advantages, this provides a common framework for expressing specifications and programs and obviates the need for elaborate mappings from specifications to programs. A potential disadvantage is that it is quite easy in Gypsy to write specifications which are semantically quite similar to the implementation.

Gypsy is descended from Pascal [11] and contains features for data abstraction, condition handling, dynamic memory management, and concurrency. The specification component of the language contains the full expressive power of the predicate calculus and the ability to write recursive functions. Specifications may be written as Floyd-Hoare style program annotations, algebraic-style axioms, or state machine descriptions.

The Gypsy Verification Environment [7] is a collection of software tools which form a development environment for creating, specifying, maintaining, and verifying Gypsy programs. These tools include a parser, verification condition generator, interactive proof checker, and algebraic simplifier. Gypsy is fully described in [6] and a methodology for using the language effectively is documented in [7].

The Z Specification Language

Key to understanding Z is the designers' "conviction that real software *can* be specified and that ordinary mathematics is the proper tool" [9]. Z purports to offer a standard mathematical notation which is "easy for a scientifically trained reader to understand; is rigorous; denotes rich concepts; and is an open notation, because you can enlarge it at will" [9].

The basic structuring concept is the *schema* [24]. A schema is an association of variable declarations and *observations* about those variables. An observation merely expresses some relation among variables. An observation can be viewed as placing a constraint upon any implementation of the specification. Schemata can be written in either a tabular or linear form; the tabular form seems to be the preferred form.

Consider the following schema for a portion of a specification of a symbol table abstraction [9].

```
___LOOKUP_______________________
| st, st' : ST
| s?  : SYM
| v!  : VAL
|______________
|
| st' = st
| s? ∈ dom (st)
| v! = st (s?)
|_______________________________
```

The top portion of the schema defines a collection of variables: a variable **st** which is a mapping from symbols (**SYM**) to values (**VAL**), an input variable **s?**, and an output variable **v!** of the indicated types. The intelligibility of the schema relies heavily upon certain conventions. The primed variable **st'** is assumed to be the final value of **st**; variables suffixed with "?" and "!" are assumed to be for input and output, respectively.

The bottom portion of the schema is a collection of *observations* stating relations among the variables. These are merely predicate calculus expressions involving the variables of the schema and may involve any of the standard operations of predicate calculus, elementary set theory, or mathematics. A

[3] We devote somewhat less attention to the description of Gypsy than to Z since most security researchers have at least a passing acquaintance with Gypsy.

standard notation is defined in [9]. In the example above, the observations indicate that the state variable is unchanged by the **LOOKUP** operation. A precondition of the operation is that input symbol **s?** is in the domain of the state mapping. The output value **v!** is the result of accessing the value currently associated with **s?** in the state. Notice that the specification is highly nonprocedural and places no constaints on the implementation other than the logical consequences of the observations.

The top portion may also include the names of other schemata indicating that these are to be included as subparts of the current schema. Common variables are shared and the collections of observations are conjoined. Inclusion is the simplest operation in a schema calculus which permits building up complex and well structured specifications by defining and combining schemata. The schema calculus provides a notation for expressing complex schemata compactly; the schema operations seem to be entirely eliminable in favor of a (possibly quite large) list of variable declarations and observations.

The Unix File System Example

A specification in Z of part of the functionality of the Unix filing system is given in [18]. The system modeled is UNIX Level 6. Operations covered include nine system calls—**read**, **write**, **create**, **seek**, **open**, **close**, **fstat**, **link**, and **unlink**—and the commands **ls** and **move**. The specification seems to have been intended as a tutorial example of the use of Z and proceeds by defining a series of progressively more elaborate mechanisms for accessing and manipulating files. At each level, additional complexity is added by defining new schemata from variable declarations, observations and previous schemata using logic and the schema calculus.

We developed a specification of the same functionality in Gypsy trying to follow as closely as possible the development style while still constructing a reasonable Gypsy specification. Our approach was to define a sequence of Gypsy scopes reflecting the added functionality at each step in the development of the specification. For example, the authors of the Z specification define an initial mechanism for reading and storing files considered as byte sequences. They then elaborate this into a mechanism for reading and storing files with a level of indirection representing the filing system. To mimic this structure, we first wrote a Gypsy scope modeling the reading and storing of files as byte sequences. A subsequent scope used the types and procedures defined there to define other procedures adding the level of indirection. Our goal was a Gypsy specification which would be as easy as possible to compare to the Z specification. In this section we illustrate the two specifications by considering the successive elaborations of **WRITE** operation.

The Basic Types

The Z Version: Types in Z seem to be rather informal; we merely declare, for example, that we want a type BYTE. We can declare sets, sequences, tuples, bags, relations, mappings, and functions. We can state whether functions are partial or total and (with a lambda expression) how they are computed. For our purposes we need the types

```
BYTE

FILE ≡ seq BYTE
```

We'll also need the naturals, but these are primitive. A constant **ZERO** of type **BYTE** is required, but there seems to be no need to explicitly define it.

The Gypsy Version: The typing mechanism in Gypsy is more formal and more restrictive. Basic types such as **INTEGER** and **BOOLEAN** are available as are the static type compositions arrays and records and the dynamic type compositions of sets, sequences, and mapping. Gypsy also has buffer types for communication between concurrent processes. The Gypsy analog of the Z types above is:

```
type BYTE = pending;

type BYTE_SEQ = sequence of byte;

const ZERO: byte := pending;

type FILE = byte_seq;
```

The keyword **pending** in Gypsy is a conceptual place-holder which makes no commitment to the ultimate implementation.

Defining the natural numbers as a type in Gypsy is awkward. Whereas the integers are unbounded in Gypsy, there is no satisfactory way to specify an unbounded subset of them. The naturals are characterized in our specification as the collection of integers between 0 and some unspecified constant. The lemma (axiom) **MAX_NAT_POSITIVE** guarantees that this range is nonempty.

```
const MAX_NAT : integer := pending;

lemma MAX_NAT_POSITIVE =
        MAX_NAT > 0;

type NATURAL = integer [0..MAX_NAT];
```

This characterization of the naturals in Gypsy is clumsy. It is likely that for any language which is mechanically processed (as Gypsy is) there will be interesting concepts which cannot be formalized conveniently within a fixed notational framework. A language such as Z which is more freely extensible seems to have a distinct advantage in this regard. On the other hand, many would argue that the desire to include too much in a formal language is exactly the cause of complexity and inelegance in languages such as PL/I and ADA. [10]

Writing Files

The Z Version: The operation of writing a file is defined in the Z specification by the schema:

writeFILE

file, file' : FILE

offset? : $\mathbb{N}$

data? : seq BYTE

$$file' = zero_{offset?} \oplus file \oplus (data?^{\circ}pred^{offset?})$$

$$\text{where } zero_k = (\lambda n{:}\mathbb{N} \mid 1 \le n \le k \bullet ZERO)$$

The uninitiated may find this specification rather daunting. Actually, it is quite simple once the meaning of the operators is understood.

zero$_k$ is a sequence of length **k** all of whose bytes are **ZERO**. ⊕ is the function overriding operator; **f⊕g(x)** equals **g(x)** unless **g(x)** is undefined, in which case it equals **f(x)**. **pred** is the predecessor function. The specification states that to determine the value of any byte in the written file one must look first at the written data, then at the previous contents of the file, and finally to **ZERO**.

Notice again the very nonprocedural nature of the specification. There is no prescription *how* the final value of file is obtained, only of *what* the final value must be.

The Gypsy Version: It is possible in Gypsy to state the specification corresponding to the Z schema **writeFILE** in a very abstract fashion essentially as a functional relationship between the input and output values of the file. This might be expressed as:

```
function WRITE_TO_SEQUENCE (in_file : file;
                             offset : natural;
                               data : byte_seq) : byte_seq =
begin
   exit (result =
            if (offset le size (in_file))
               then in_file [1..offset-1] @ data
               else in_file @ n_zeros ([offset - size (in_file)] + 1)
                             @ data
            fi);
end; {write_to_sequence}
```

```
function N_ZEROS (n: integer): byte_seq =
begin
   exit (  result
         = if n le 0
       then null (byte_seq)
       else zero :> n_zeros (n - 1)
            fi);
   pending;
end; {n_zeros}
```

The Gypsy version is somewhat more verbose but quite similar in spirit to the Z version. Preference and experience determines which is more daunting. Notice the need in Gypsy to declare the auxiliary function **N_ZEROS** comparable to the Z construct $\mathbf{zero}_k$.

The Gypsy function **WRITE_TO_SEQUENCE** defines the desired input/output relation of the **WRITE** operation, but it is not the operation itself. It is natural in Gypsy to characterize the operation itself as a function or procedure and use the specification function **WRITE_TO_SEQUENCE** to state a constraint on its behavior.[4]

```
procedure WRITE_FILE (var fl : file;
                       offset : natural;
                         data : byte_seq) =
begin
   exit fl = write_to_sequence (fl', offset, data);
   pending;
end; {write_file}
```

This has essentially the same content as the Z schema, where the **exit** specification gives a postcondition of the routine analogous to the Z schema observation. The presence of the keyword **pending** in place of the procedure body indicates that no commitment is currently made to an implementation. Any implementation supplied later must satisfy the specification.

File Storage

The Z Version: In the file system we access a file via its *file_id*, a number supplied by the system when the file is created. This implies a mapping from file_id to files specified in Z by a new type **FID** and a schema.

```
 ___SS________________________
|  fstore : FID ⇸ FILE
|_____________________________
```

A concept that recurs in several operations on stored files is the notion of accessing the file within the store. This is expressed in the following *framing* schema:

[4] Primed variables such as fl' represent the *input* values of variable parameters in Gypsy as opposed to the *output* values in Z.

$$\begin{array}{|l}
\text{\_\_\_}\phi SS\text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
SS, SS' \\
file, file' : FILE \\
fid : FID \\
\hline
file = fstore\ (fid) \\
fstore' = fstore \oplus \{fid \rightarrow file'\} \\
\hline
\end{array}$$

In traditional mathematical parlance this schema might translate as: *let fstore be a storage system in which file is associated with fid.* Notice that we specify explicitly how **file** is computed; thus in an expansion of this schema, we could replace all occurences of **file** by **fstore (fid)**. This notion is called *hiding* of the variable in Z.

Given the framing schema ϕ**SS**, the notion of writing a file in the storage system can be expressed in quite a terse fashion as:

$$\begin{array}{|l}
\text{\_\_\_}writeSS\text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\phi SS \\
writeFILE \\
\hline
\end{array}$$

If desired, the expansion of this schema can be obtained by conjoining its constituent schemata. Common variables and observations are recorded only once. The result is

$$\begin{array}{|l}
\text{\_\_\_}writeSS\text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
SS, SS' \\
fid : FID \\
file, file' : FILE \\
offset? : \mathbf{N} \\
data? : seq\ BYTE \\
\hline
file = fstore\ (fid) \\
file' = zero_{offset?} \oplus file \oplus (data?^{c}pred^{offset?}) \\
fstore' = fstore \oplus \{fid \rightarrow file'\} \\
\\
\text{where } zero_{k} = (\lambda n{:}\mathbf{N} \mid 1 \leq n \leq k \bullet ZERO) \\
\hline
\end{array}$$

which may be simplified using any of the rules of predicate calculus.

The Gypsy Version: These same concepts may be expressed straightforwardly in Gypsy. The types we need are declared as follows:

```
type FILE_ID = pending;

type STORAGE_SYSTEM = mapping from file_id to file;
```

The functionality of **writeSS** is expressed in the following Gypsy procedure:

```
procedure WRITE_STORED_FILE (var SS : storage_system;
                                 FID : file_id;
                                data : byte_seq;
                              offset : natural) =
begin
   entry FID in domain (SS');
   exit  SS = SS' with (into [FID]
                     := write_to_sequence (SS'[FID], offset, data));
   write_file (SS[FID], offset, data);
end; {write_stored_file}
```

Notice that this procedure makes use of the earlier version just as the Z schema made use of its predecessor. The specification is the exact analog of that for **WRITE_FILE**, with the change made to a component of the file structure rather than to the file in isolation. This is the Gypsy counterpart of the Z notion of "hiding" discussed above. The key difference between the Gypsy and Z versions is that in the Gypsy code we have procedural abstraction rather than the schema abstraction of Z.

In this case we give a body to the procedure rather than leaving it **pending**. It seemed natural to do so for two reasons. It illustrates that **WRITE_STORED_FILE** is merely a specific instance of **WRITE_FILE**, where the file var parameter is obtained via indirection through its file_id. Also it allows us to *prove* the correctness of this procedure assuming the correctness of **WRITE_FILE**.

There is a subtlety in the Z specification which becomes more explicit in the Gypsy version. In the Z version, **fstore** is declared as a partial function; the observation **file = fstore (fid)** assures that it is defined at **fid**. This is stated explicitly in the entry specification on the Gypsy routine.

Sequential Access to Files

The next step in the development is to add the notion of sequential access to files via *channels*. A channel records an association between a file id and a current position in the file. Sequential access in the file commences from that position.

The Z Version: This association is made in Z with the schema:

```
___CHAN________________________
| fid : FID
| posn : N
|______________________________
```

An additional important property of channels is that the **fid** of the channel never changes, expressed by:

```
___ΔCHAN_______________________
| CHAN, CHAN'
|_____________
|
| fid' = fid
|______________________________
```

The operation of writing a file via a channel makes use of the previous schemata **writeSS** and ΔCHAN along with some observations to characterize the result.

```
___writeCHAN___________________
| writeSS
| ΔCHAN
|_____________
|
| offset? = posn
| posn' = posn + #data?
|______________________________
```

Here the # operator returns the length of its argument. Notice that the parameter **offset?** to schema **writeSS** is supplied by the **posn** component of the channel.

Finally, we wish to add a named system of channels for performing sequential access. We add the data type **CID** of channel ids and a mapping from channel ids to channels.

```
___CS______________________________
| cstore: CID ⇸ CHAN
|__________________________________
```

We need also a schema describing the writing of a channel accessed via the channel system. This is merely an instance of the **writeCHAN** schema with **posn** supplied from the channel store.

```
___writeCS_________________________
| CS, CS'
| writeCHAN
|______________
|
| posn = cstore (fid)
|__________________________________
```

<u>The Gypsy Version:</u> The most natural way in Gypsy of associating two dissimilar pieces of data is a record structure. We could have defined a **CHANNEL** record type of two fields. In the **writeCHAN** operation, this would be convenient. However, looking ahead to the use we'll make of channels, it seemed that this way of structuring would be inconvenient.

This illustrates a characteristic difference between Gypsy and Z. In Z, the association of data within a schema has no connotations for an implementation structuring. Individual pieces of data can be aggregated into various different schemas. In Gypsy, on the other hand, associating data items into a structure, a record for example, makes it very difficult to re-associate those data items differently at a later point.

Our declarations and the definition of the operation for writing a file via a channel are specified as follows:

```
type POSITION = natural;

type CHANNEL_ID = file_id;

procedure WRITE_CHANNEL (var SS : storage_system;
                            data : byte_seq;
                         chan_id : channel_id;
                            var posn : position) =
begin
   entry chan_id in domain (SS);
   exit  SS = SS' with ([chan_id]
                  := write_to_sequence (SS[chan_id], posn', data))
       & posn = posn' + size (data);
   write_stored_file (SS, chan_id, data, posn);
   posn := posn + size (data);
end; {read_channel}
```

Because we did not want to create a single data structure representing the channel, it was necessary to pass the channel_id and position as separate parameters. This has an associated benefit of guaranteeing syntactically that the channel_id parameter could not be altered by the procedure invocation as called for in the Z schema **ΔCHAN** since it is a *const* rather than a *var* parameter. If the channel id and position parameters had been fields in a record parameter to this routine, it would have been more difficult to assert that **WRITE_CHANNEL** does not alter the channel id.

We now define the channel system as the following mapping:

```
type CHANNEL_SYSTEM = mapping from channel_id to position;
```

The operation which allows us to write a file sequentially using the channel system is coded in Gypsy as follows:

```
procedure WRITE_CS ( var SS : storage_system;
                     var CS : channel_system;
                    chan_id : channel_id;
                       data : byte_seq) =
begin
   entry  chan_id in domain (CS)
        & chan_id in domain (SS);
   exit   SS = SS' with ([chan_id]
                := write_to_sequence (SS'[chan_id], CS'[chan_id], data))
        & CS = CS' with ([chan_id] := CS'[chan_id] + size (data));
   write_channel (SS, data, chan_id, CS[chan_id]);
end; {write_CS}
```

It is necessary to pass in both the storage system and channel system since both the file and the current position are updated by the **WRITE_CS** operation. Notice also that we need to assure in the entry specification that the channel_id is a proper file_id in the file system. We'll address this issue again in the next section.

The Access System

The last component of the specification we'll consider is the access system. The access system is merely the combination of the storage and the channel system. However, we want to assure that no channel contains a file id for which there is no associated file in the storage system.

The Z Version: This is expressed in Z by the following schema.

```
___AS________________________________
|   SS
|   CS
|
|_______________
|
|   ran (fid∘cstore) ⊆ dom fstore
|_______________________________________
```

The observation in this schema expresses an invariant which must be preserved by every operation on the access system. Since the schema abstraction is nonprocedural, the invariant is simply inherited by every schema which uses **AS**.

The write operation using the access system is specified using the framing schema

```
___ϕAS_______________________________
|   AS, AS'
|   ΔCHAN
|   cid : CID
|
|_______________
|
|   CHAN = cstore cid
|   cstore' = cstore ⊕ {cid ↦ CHAN'}
|_______________________________________
```

and the following combination

```
___writeAS___________________________
|   ϕAS
|   writeCHAN
|_______________________________________
```

The Gypsy Version: The desire to associate an invariant with a collection of data structures leads naturally in Gypsy to the abstract data typing facility. In this case the data type represents the aggregate of the storage system and the channel system.

```
type ACCESS_SYSTEM <SS, set_SS, ...> =
   begin
      AS: record (SS: storage_system;
                  CS: channel_system);
      HOLD domain (AS.CS) sub domain (AS.SS);
   end;
```

The type **ACCESS_SYSTEM** is a Gypsy abstract data type. The abstract typing mechanism in Gypsy serves two distinct purposes: to hide the implementation of a type and to permit the association of an invariant with the type in the form of the **HOLD** specification. The *access control list* including **SS**, **set_SS**, and possibly others gives the list of routines which are permitted access to the concrete (record) structure of the type. Each of these must be proved to maintain the invariant.

The use of Gypsy data abstraction for our example is somewhat unfortunate because we are really concerned only with maintaining the invariant; the data hiding aspect of abstract typing is primarily a nuisance in the current context. Since we will want to access the various components of the access system, it is necessary to write functions which will permit us to access and set components. For the storage system component, such functions are:

```
function SS (AS: access_system): storage_system =
begin
   cexit result = AS.SS;
   result := AS.SS;
end; {SS}

procedure set_SS (var AS : access_system;
                      SS : storage_system) =
begin
   cexit AS = AS' with (.SS := SS);
   AS.SS := SS;
end; {set_SS}
```

We would have similar functions for the channel system component of the abstract type. It is also necessary to define a special function which characterizes equality for the abstract type.

It is syntactically disallowed for any routine to refer to the concrete structure of the abstract type except those routines mentioned on the access control list. Even these cannot refer to the concrete structure in their abstract external specifications (**entry** and **exit**). The **centry** and **cexit** specifications of these routines may refer to the structure, but they are visible only in proof contexts in which the concrete structure of the type is visible. Thus **SS** and **set_SS** are abstract accessors which must be used in most contexts in place of direct access to the **SS** component of the record structure.

The write operation using the access system then becomes

```
procedure WRITE_AS ( var AS : access_system;
                    chan_id : channel_id;
                       data : byte_seq) =
begin
   entry  chan_id in domain (CS (AS));
   exit  SS(AS) = SS(AS) with ([chan_id]
                 := write_to_sequence (SS(AS)[chan_id],
                                       CS(AS)[chan_id], data))
        & CS(AS) = CS(AS) with ([chan_id]
                                 := CS(AS)[chan_id] + size (data));
   write_channel (SS(AS), data, chan_id, CS(AS)[chan_id]);
end; {write_AS}
```

The astute observer will have already noted that this is exactly the definition of **WRITE_CS** with adjustments made for the abstract data typing and the combination of CS and SS parameters into one structure.

Though we stop our exposition of the specifications here, the interested reader is invited to investigate the complete specifications. The Z version of the full spec is described in [18]. Our Gypsy version is available upon request.

Comparing the Specifications

Our investigation of the UNIX Filing System example has highlighted various features of the two specification languages. Though the resulting specifications are superficially quite different, we have attempted to point out the underlying similarities. We would aver that both specification languages can result in elegant readable specifications if used with skill and care. There are distinct differences, though, which are worthy of note.

Expressiveness of the Languages

The fact that Gypsy is an implemented language means that there are certain constraints upon the expressiveness of the language imposed by the parser. Gypsy contains the full first order predicate calculus and the ability to define functions recursively. In addition, there is an extensive collection of data types including sets, sequences, and mappings. However, we have seen that it is sometimes awkward to express certain concepts—the natural numbers are a good example—in a satisfactory fashion. Also, the lack of polymorphism in Gypsy means that it is often necessary to write very similar functions to perform analogous operations on, say, sequences of integers and sequences of Booleans. Many concepts which might be desirable from a programming standpoint—pointers, floating point, global variables, functions with side effects—are explicitly excluded because of the difficulties they present for verification.

Z suffers from no such lack of expressiveness. In addition to the huge notational variety suggested by the language designers [9], the writer of Z specifications is free to use or invent notation at will. This gives the Z user freedom to write the cleanest specifications possible.

On the other hand, the free and easy use of notation in Z may have some disadvantages. The semantics of Z is inherited from mathematics and in that respect well defined. However, this is an extremely powerful underlying theory. There is certainly no way to insure that Z specifications are realizable or even consistent. The same can be said of Gypsy specifications, though the procedural nature of the language imposes a bias toward constructive specifications. Z specs are often highly non-constructive.

Also, much of the readability of a Z spec derives from various notational conventions: the suffixes distinguishing input and output variables, for example. Since these are not enforced by a language processor it is possible to violate them quite easily.

Structuring of the Specifications

The primary structuring concept in Z is the schema abstraction; in Gypsy it is procedural abstraction. Either permits a well structured development style and a clean modular specification. Z seems to win for sheer brevity and abstractness of the resulting specifications.

Much of the verbosity of the Gypsy specs comes from Gypsy's proscription of non-local referencing. This requires that all data structures accessed within a module be either local or passed in as parameters and this tends to clutter the procedure header. However, this has the strong advantage that Gypsy modules can be understood in isolation from their calling environments and that the effects of a Gypsy procedure are very strictly constrained. A constraint on the language called the *Independence Principle* assures that any module is analyzable/provable with regard only to its own code and the external specifications of any routines it calls.

Fully understanding a Z spec may require expanding all of the schemata in the tree of schema definitions defining it. This could be formidable indeed. The use of schema naming conventions seems to make this seldom necessary. However, again this requires that the specifier follow the standard conventions.

Data structuring in Z is much less constraining than in Gypsy. Consider our formalization of the Unix *access system*. By encapsulating the storage system and channel system into a single abstract data type, we conceptually bind them together. In a later context (not discussed above) it is convenient to consider a combination of the storage system with some other combinations of systems (not including the channel system). In Z, this means simply including the appropriate schemata; including one schema within another does not "hide" it from any future uses. In Gypsy, there is a conceptual structural mismatch arising from considering the "same" system component as a piece of two different aggregations.

Procedural vs. Nonprocedural

The debate over the relative merits of procedural versus non-procedural specifications at times takes on an almost religious fervor. Suffice it to say that Z is highly non-procedural. Gypsy specifications can be non-procedural though this is typically not the most natural style. The advantage of non-procedural specifications is that they are largely implementation independent. This is evident from the Z file system example. The danger of non-procedural specifications is the lack of assurance that they are realizable.

The ability to supply bodies to our Gypsy procedures proved seductive and insofar as these procedure bodies are considered as part of the specification, they constrain allowable implementations. On the other hand, the advantages of executable specifications for rapid prototyping have often been cited. Gypsy specifications are not executable because of the current lack of an interpreter for the language. It would not be difficult to supply one, however.[5] Also, some aspects of the Gypsy specification language are intended for *run-time validation*, the evaluation of specification expressions during program execution. This allows the checking of specifications very directly against the run-time behavior of a system.

Amenability to Code Level Specification

A difference in these specification styles which may be particularly relevant to secure system development efforts is the applicability to specifications at or near the code level. One of the increasing demands upon verification technology alluded to earlier is the demand to close the gap between the specification level and the machine code implementation of the system running on actual hardware. This is evident particularly in the requirements of the "beyond A1" certification level of DoD's *Trusted Computer Systems Evaluation Criteria* [3].

Z has been used for specifying some "real" software systems of impressive size including one system of over 80,000 lines of code at IBM-Hursley. There is no reason why a specification in Z cannot be as detailed and as near the code level as is required. There is also ongoing research into refining Z specifications into code in a guarded command language [17]. Presumably this could then be translated into C or other suitable implementation language.

Gypsy has been used for code level specification and verification on several projects. [21, 19] For these projects Gypsy was used as the implementation language and mechanically translated into Bliss which was then compiled. [20] Most current uses of Gypsy in the development of secure system applications, however, have been for specification at the design level. There is currently no Gypsy compiler available except a prototype verified compiler for a very small subset of the language [25]. The result is that Gypsy design level specifications are translated by hand into C or some other suitable implementation language, an error-prone process [26].

Arguably, Gypsy has an advantage over Z in this process in that there is a clearer mapping from procedural Gypsy code to an implementation than from a non-procedural Z specification to an implementation. However, it may also be that the procedural nature of a Gypsy specification obscures rather than clarifies the mapping if the implementation is structurally quite different from the spec.

[5] It might be possible to supply some such notion for Z as well. Logic programming gives a procedural as well as a declarative interpretation to logical formulae. However, Z is a much richer language than typical logic programming languages.

Mechanical Support

The clearest distinction between Gypsy and Z is in the area of mechanical support for language processing. There currently seems to be no language processing capability for Z. The Gypsy Verification Environment (GVE), on the other hand, is a mature and well integrated collection of software tools for developing and processing Gypsy programs and specifications. These tools include a parser, database system, verification condition generator, interactive proof checker, and algebraic simplifier.

A mechanical parser is particularly beneficial from the point of view of writing consistent specifications. We noticed in studying the Z Unix File System specification [18] that there was a least one schema (`createCS`) which is referenced but never defined.[6] This sort of oversight is very easy to eliminate with mechanical parsing.

Proofs about the Specifications

For both languages, it is possible to do proofs about the specifications. For Z, this follows from the fact that the specification language in some senses just *is* elementary mathematics. Users interested in doing proofs will find themselves on the safe and sure ground of elementary mathematics. Proof, however, does not seem to be a high priority for Z users. Possibly this is because the focus of Z use is on *specification* of software systems, not on formal verification which tends to focus on proving the conformity of specifications and code.

The Gypsy system is very heavily oriented toward proofs of correctness. An overriding design criterion for the language was that every construct have associated proof rules. The verification condition generator processes programs annotated with assertions to generate verification conditions (VC's) adequate to assure the conformity of code and specifications. These verification conditions are conjectures which can be proved using the Gypsy interactive theorem prover. It is also possible to state and prove *lemmas*.

The need to resort to the process of verification condition generation for Gypsy programs is due to the procedural nature of the language. It is sometimes argued that the process of VC generation obfuscates the relation between specifications and code. The VC's often bear little obvious relation to the code. However, this seems to be a necessary price for having procedural constructs in the language. It is possible to reason about procedural programs directly with respect to a formal semantics, but it is much more difficult to do so [25].

Conclusions

We have compared and contrasted two specification languages—Gypsy and Z—in light of a common example. Each provided some obvious advantages and disadvantages.

Z allows the construction of very clear and elegant specifications. It has been used with good results in specifying large software systems. The principle failings of the language and its current usage seem to be the following.

1. The expressive freedom allowed by the language can, if abused, easily result in specifications which are either not satisfiable or for which there is no efficient implementation.
2. Because of the highly non-procedural character of specifications in Z, there may be no clear mapping from specification to implementation. Thus it might be very difficult to construct a believable specification to code correspondence argument.
3. The greatest failing of Z currently seems to be the lack of mechanical support for language processing. Inconsistencies and gaps in the specifications could be easily eliminated by a parser.

Gypsy is a combined specification and programming language with extensive software support. The following comments can be made about Gypsy and its implementation.

1. Some common mathematical notions are difficult to express in Gypsy. We noted the natural number

[6] Interestingly enough, this is actually evident from the index of schemas and components given by the authors. Schema `createCS` is listed as being used within schema `create`; there is no separate entry for the *definition* of `createCS`.

data type as an example; because of the absence of pointers, trees are also awkward to express. However, it is not quite accurate to say that Gypsy is uniformly less expressive than Z. It is unclear, for example, how difficult it would be to specify in Z concurrent programs which Gypsy allows.[7]

2. The mechanisms of abstraction in Gypsy—procedural and data abstraction —are less flexible than schema abstraction. In particular, it is difficult to associate components of a system into various *different* aggregations.

3. Schema based specifications tend to be more succinct and abstract than the procedural specifications of Gypsy. This can be interpreted as implying that the procedural nature of Gypsy specifications imposes unnecessary constraints on an implementation.

Our experience in comparing Gypsy and Z leads us to believe that the relative strengths of the two specification languages are in fact quite complementary. The major failing of Z—the lack of mechanized language support—is also the easiest to remedy. The lessons learned in the development of the Gypsy Verification Environment could serve as a model for the development of a mechanical support environment for Z.

Because of its procedural constructs and strong mechanical support for proofs about programs, Gypsy will likely continue to have the edge over a language like Z in secure system development efforts. But the lessons gained by comparing these very different specification paradigms may inform future changes and improvements in both languages and their support environments.

Acknowledgments

Thanks to Matt Kaufmann and to the reviewers for some quite insightful comments on an earlier version. This work was supported in part at Computational Logic, Inc., by the Defense Advanced Research Projects Agency, ARPA Orders 6082 and 9151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

References

1. W.E. Boebert, W.D. Young, R.Y. Kain, S.A. Hansohn. Secure ADA Target: Issues, System Design, and Verification. *Proc. Symposium on Security and Privacy*, IEEE, 1985.

2. M. Cheheyl, M. Gasser, G. Huff, J. Millen. "Verifying Security". *ACM Computing Surveys 13*, 3 (September 1981), 279-340.

3. Department of Defense. *Trusted Computer Systems Evaluation Criteria*. DOD 5200.28-STD, December, 1985.

4. Bill Flinn and Ib Holm Sorensen. CAVIAR: A Case Study in Specification. In Ian Hayes, Ed., *Specification Case Studies*, Prentice-Hall, Englewood Cliffs, N.J., 1987, pp. 141-188.

5. D.I. Good. SCOMP Trusted Processes. ICSCA Internal Note 138, The University of Texas at Austin.

6. D.I. Good, R.L. Akers, L.M. Smith. Report on Gypsy 2.05. Tech. Rept. ICSCA-CMP-48, Institute for Computer Science and Computing Applications, The University of Texas at Austin, February, 1986.

7. D.I. Good, B.L. Divito, M.K. Smith. Using The Gypsy Methodology. Institute for Computing Science, University of Texas at Austin, June, 1984.

8. D.I. Good, A.E. Siebert, L.M. Smith. Message Flow Modulator Final Report. Tech. Rept. ICSCA-CMP-34, Institute for Computing Science, University of Texas at Austin, December, 1982.

[7] There is apparently ongoing research on specifying concurrent programs in Z based on the approach of [12]

9. Ian Hayes (editor). *Specification Case Studies*. Prentice-Hall, Englewood Cliffs, N.J., 1987.

10. C.A.R. Hoare. "The Emperor's Old Clothes: 1980 Turing Award Lecture". *Communications of the ACM 24*, 2 (February 1981), 75-83.

11. K. Jensen and N. Wirth. *Pascal User Manual and Report*. Springer-Verlag, 1974.

12. C.B. Jones. "Tentative Steps Toward a Development Method for Interfering Programs". *ACM Trans. on Programming Languages and Systems 5*, 4 (October 1983), 596-619.

13. Matt Kaufmann and W.D. Young. Comparing Specification Paradigms for Secure Systems: Gypsy and the Boyer-Moore Logic. Proceedings of the 10th National Computer Security Conference, National Bureau of Standards, September, 1987.

14. J. Keeton-Williams, S.R. Ames, B.A. Hartman, and R.C. Tyler. Verification of the ACCAT-Guard Downgrade Trusted Process. Tech. Rept. NTR-8463, The Mitre Corporation, Bedford, MA., 1982.

15. Richard Kemmerer. Verification Assessment Study Final Report. In 5 volumes, unpublished.

16. Carl E. Landwehr. "The Best Available Technologies for Computer Security". *IEEE Computer 16*, 7 (July 1983), 86-100.

17. C. Morgan, K. Robinson, P. Gardiner. On the Refinement Calculus. Draft, July 1988.

18. Carroll Morgan and Bernard Sufrin. Specification of the UNIX Filing System. In Ian Hayes, Ed., *Specification Case Studies*, Prentice-Hall, Englewood Cliffs, N.J., 1987, pp. 91-140.

19. A.E. Siebert and D.I. Good. General Message Flow Modulator. Tech. Rept. ICSCA-CMP-42, Institute for Computing Science, University of Texas at Austin, March, 1984.

20. L.M. Smith. Compiling from the Gypsy Verification Environment. Tech. Rept. ICSCA-CMP-20, Institute for Computing Science, The University of Texas at Austin, August, 1980.

21. M.K. Smith, A. Siebert, B. Divito, and D. Good. "A Verified Encrypted Packet Interface". *Software Engineering Notes 6*, 3 (July 1981).

22. J.M. Spivey. *Understanding Z: a Specification Language and its Formal Semantics*. Cambridge University Press, 1988.

23. Bernard Sufrin. Towards a Formal Specification of the ICL Data Dictionary. In Ian Hayes, Ed., *Specification Case Studies*, Prentice-Hall, Englewood Cliffs, N.J., 1987, pp. 189-217.

24. J.C.P. Woodcock. Structuring Specifications: Notes on the Schema Notation. Oxford University Computing Laboratory, August, 1987.

25. W.D. Young. A Verified Code Generator for a Subset of Gypsy. Tech. Rept. CLI-33, CLInc, November, 1988.

26. W.D. Young, J. McHugh. Coding for a Believable Specification to Implementation Mapping. Proceedings of the 1987 Symposium on Security and Privacy, IEEE, 1987.

EVALUATION OF SECURITY MODEL RULE BASES

John Page, Jody Heaney, Marc Adkins[1], Gary Dolsen

Planning Research Corporation
Government Information Systems
1500 Planning Research Drive
McLean, VA 22102

ABSTRACT

The findings presented in this paper are results of a contract effort to establish the feasibility of rule-based TCB's for SDIO and RADC [1].[2] A TCB with interchangeable rule bases would be able to respond to changes in policy or military readiness without a redesign of the kernel, and would promote the maintenance of secure systems. The initial phases of the effort covered the derivation of rule bases for three computer security models: the Bell and LaPadula model, the Military Message System model, and the MAC portion of the SeaView Relational DBMS model. The derivation process was automated to a large extent by the Security Model Development Environment (SMDE) which was developed as part of this effort. While the primary purpose of the SMDE is to generate and exercise model rule bases, its tools provided highly useful information about the models themselves. The end result was a new way of viewing and analyzing security models by observing their rule bases.

INTRODUCTION

The primary aim of this evaluation was to view three different security models from the common point of reference provided by the Security Model Development Environment (SMDE) prototype. This form of analysis differs from many current practices since it places a strong emphasis on the functional aspects of the model represented by operations and rules. As it is important for the reader to understand the context in which these observations are made, a brief summary of the SMDE prototype is offered. Results for each of the three models are presented in turn. While common themes may be seen from model to model, the material represented herein has been selected to portray the scope and breadth of model evaluation with the SMDE prototype. Although it is hoped that the reader will have some familiarity with the models, supplementary material is provided to re-acquaint the reader with key model features. It was also necessary to provide a number of interpretations to adapt the models into the SMDE format, and these are identified as they appear. A full description of these activities is found in [2].

CREATING RULE BASES FOR SECURITY MODELS

It is first necessary to define what we mean by a rule base. A rule base is a collection of rules that can be mapped one-to-one onto the set of operations the model is to support, much like that found in the Bell and LaPadula (BLP) model. Each rule is a pre-check for a given operation, specifying what conditions must hold in order for the operation to execute without breaching the security of the system. When presented with an operation request, the kernel control mechanism will evaluate the rule for the operation. The operation is then allowed to proceed if the rule is satisfied. Otherwise, the operation is disallowed and any appropriate action will be taken.

Not all models are necessarily suitable for the derivation of model rule bases. The methodology underlying the process assumes that the model is based on an inductive state machine schema of secure states and secure transitions, like the one proposed in the BLP model. More abstract models would need to be decomposed into a state machine representation. The operational aspects of the model to be represented must also be explicitly stated. If a model does not provide a set of operations, a representative set must be defined. The SMDE process of deriving and

[1] This author's current address is Booz-Allen & Hamilton, Inc., 4330 East West Highway, Bethesda, MD 20814.

[2] Contract number F30602-86-C-0190 for the Strategic Defense Initiative Office and Rome Air Development Center.

evaluating the rule base for a given model requires several steps, as shown in Figure 1. The process starts with a paper model which is expressed in the Common Notation, a machine processible format. The model expression is parsed by the Model Translator and a rule base is derived by the Rule Generator. The rule base can then be exercised in the Testbed to determine how the components of the model interact. Each of these steps is described in greater detail in the following paragraphs.

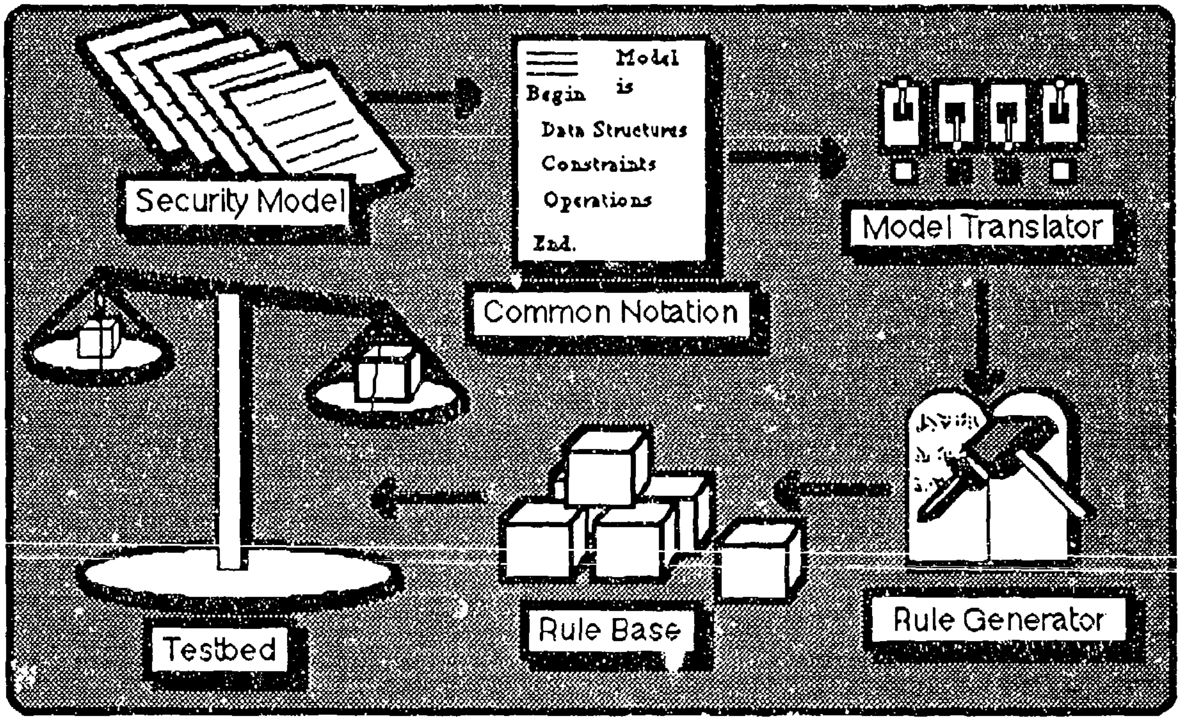


Figure 1. The Derivation Process

The model must first be represented in the Common Notation, a notation for the expression of computer security models developed for the SMDE and described in [3]. A full representation of a model in the notation consists of three major components: data structures, constraints, and operations. *Data structures* are declared in type and variable declarations common to many programming languages. The example below illustrates the sequence of type declarations necessary to define the current access matrix for the BLP model:

```
--- type declarations for current access matrix
type Accesses   is (read, write, append, execute);
type Access_set is set of Accesses;
type Access_matrices is
   Array from Subjects, Objects to Access_set;
--- variable declaration for current access matrix
current_access : Access_matrices;
```

Constraints represent the security properties of the model. Constraints may take one of two different forms, static or dynamic. Static constraints are state invariants, such as the Simple Security Property, which must hold on all data structures in a given state for that state to be secure. This property of the BLP model is represented in the Common Notation as:

```
static constraint Simple_Security_Property is
begin
  --- for all subjects and objects it must be true that
    for all sub : Subjects; ob : Objects |
  --- current read or write access between a subject and
  --- an object implies that
      ( read  in current_access(sub,ob) or
       write in current_access(sub,ob)     ) -->
  --- the current security label of the subject dominates the object
          current_security_label(sub) >=
           security_label(ob);
end Simple_Security_Property;
```

Dynamic constraints correspond to properties which must be satisfied during secure transitions. (The security properties of the MMS model are phrased in this manner.) Dynamic constraints often compare the values of a variable between two different states, or refer to knowledge which is only available during the execution of an operation. An example of a dynamic constraint in the context of the BLP model is presented under the Control Attribute discussion.

Operations describe the functionality of a system based on a model. An operation is a description of the changes that the real operation's execution would make to the system state. In essence, they describe the state transition resulting from the operation. The Get_Read operation for the BLP model, for example, describes the addition of read access to the current access set:

```
operation Get_Read (user: Subjects;
                    ob:   Objects);
begin
  --- the operation adds read access to current_access(user, ob)
    current_access(user,ob) :=
      current_access(user, ob) + read;
end Get_Read;
```

Note that the operation description does not include any pre-checks to ensure that the execution of the operation will not breach the security of the system. Pre-checks are automatically created for a given model using the SMDE.

The Common Notation representation of the model is then parsed by the Model Translator Tool, and an internal representation of the model is stored in a parse tree. The next step of the derivation process involves producing a rule for each of the operations. Rules are produced by the Rule Generator, an innovative software tool which derives rules from the parsed description of the model. This complex process correlates the changes represented by the operation with the conditions specified by the constraints in order to determine what conditions must be satisfied for the operation to execute without breaching system security. For the Get_Read operation cited above, the output of the Rule Generator is:

```
function GET_READ_RULE ( user: Subjects;
                         ob  : Objects  )
           return Boolean is
begin
 -- From static constraint Simple_Security_Property:
  Dominates(current_security_label(user),
            security_label(ob)          ) and
 -- From static constraint Discretionary_Security_Property:
  Member_of(read, access_permission(user, ob);
end GET_READ_RULE;
```

This rule specifies that `user` may attain read access to `ob` if `user`'s current security level dominates that of the object and `user` has the necessary discretionary access rights.

Once rules have been generated for each of the operations supported by the model, they are stored as the rule base of the model. The final stage of the process involves loading the rule base into a specially designed Trusted Computing Base (TCB) kernel. Since one goal of our work was to investigate the feasibility of the latter, it was necessary to design a TCB Testbed which simulates the control mechanism of a TCB kernel that mediates operation requests. The Testbed allows a modeler to simulate the execution of the model rule base by executing *scenarios*. A scenario is a Common Notation program that describes an initial secure state configuration, and then presents the rule base with a series of operation requests. For each request, the Testbed invokes the rule that tests to see if the necessary conditions are satisfied to preserve the security of the system. If the rule returns true, the changes described by the operation are made to the system state, otherwise the operation is rejected.

While the SMDE is primarily designed to investigate the feasibility of rule-based TCBs, it also contributes to the analysis and evaluation of computer security models. The syntactic demands of the Common Notation require a full, explicit representation of a model. In many cases, it was necessary to flesh out intuitive assumptions within models in order to create a satisfactory rule base. These interpretations, which would be necessary in any implementation from a model, have raised many interesting issues that may not have arisen under traditional methods of model evaluation. Another distinctive aspect of the SMDE is its emphasis on the set of operations which a model is to support. While many traditional model development methodologies postpone operational specifics until implementation, our experience has shown the operational considerations as indispensable to security model design.

THE BELL AND LAPADULA RULE BASE

The BLP model, Volume 3, [4] served as a baseline in our effort to generate a functioning rule base from a security model. Not only did the model (volume 3, unless otherwise stated) provide a full suite of operations, but it also contained a manually derived set of rules to compare against the output of the rule generator. This greatly simplified the derivation of a rule base for the model. Nonetheless, our initial attempts highlighted some aspects of the model which required further exploration.

The Control Attribute

A subject controlling an object may alter its access permissions, change its security label within certain restrictions, or delete it from the object hierarchy. A control attribute is not represented explicitly within the model. It is assumed instead that if a subject has write access to the ancestor of a given object, then the subject "controls" the object. If a subject does not have write access to the object's ancestor, then the subject does not have the benefits of controlling the object.

How is the control attribute enforced within the model? There are no formal properties of the model which state that a subject must have write access to the ancestor of an object in order to modify its access permissions. Instead, this property is embedded in the BLP rules for the operations that modify access permissions, such as Grant, Create Object and Delete Object.[3] Since the SMDE derives rules automatically, to include this embedded property into the rules requires an explicit statement of the corresponding property from the original model. Therefore, it was necessary to define a constraint which stipulates that the access permissions of an object may not be modified without the subject initiating the operation having write access to the object. Two additional constraints that we defined require a subject to "control" an object in order to attempt to alter its security label or to alter the object's position in the hierarchy. All three of these constraints had to be phrased as dynamic constraints, because the properties they specified applied to state transitions as opposed to a state at rest. Since the BLP model was formulated in terms of static

[3] This control attribute discussion does not imply any incorrectness in the original model formulation. The authors' intent here is to show the Common Notation requirement for clarity of expression.

constraints (state invariants), the only way for the authors of the model to represent these properties was to splice them into the rules for the operations that were affected. Our interpretation represented the control mechanisms of the model in the form of global security properties.

Propagation of Access Permissions

It is possible for more than one subject to control an object at a given time. Since there is no numerical bound to the number of subjects which can have write access to the ancestor of a given object, this implies that there is no limit to how many subjects can control a given object at the same time. Aside from any difficulties springing from an object being simultaneously observed and modified by a variety of subjects, this notion of multiple ownership can lead to an interesting degenerate state. Suppose that an authorized but malevolent user wishes to seize control of an object that is currently shared with others. Is it possible for a subject to take advantage of the discretionary access mechanisms to usurp control privileges from the original owner of an object? What limitations do the mechanisms of the model place on such activity?

While the previous topic was investigated by direct analysis of the rule base, exploration of this issue was not best performed in this manner. In this case, it was more beneficial to simulate the behavior of the rule base in the Testbed. A scenario was written to create a system state where two subjects (one benign, and one malevolent) have control of the same object. The malevolent subject attempts to take undisputed control of the object, while the benign one tries to maintain control of the object.

The output from the Testbed revealed several interesting points. It was indeed possible for a malevolent user to seize control of the object. When two subjects are locked in battle for the possession of an object, victory goes to the subject that has write permission to the highest object in the hierarchy. Once this subject achieves write access to the highest ancestor, the subject may sweep down the tree, altering access permissions until it has sole control of the object. If the highest write permission for both subjects is to the same object, a stalemate occurs, and each side alternates taking control of the object and yielding it to the other.

The model does provide some limitations to stem this sort of activity. Since the root object has no ancestor, it cannot be controlled. Thus a subject is prevented from controlling the entire hierarchy. It also implies that the access permissions of the root may never be altered during the course of the system's history, given this interpretation of the control attribute. The downward spread of control is confined by security levels, since a subject could never get control of an object whose ancestor had a higher security level than his maximum level. Therefore, a subject may not always be able to extend its control down to the bottom of the hierarchy.

Object Deletion

Deriving a rule for the Delete Object operation also raised some thought-provoking issues about the control attribute. If a subject controls an object, it is entitled to delete the object. If the object is a leaf node in the hierarchy, this creates few difficulties. All accesses to the object are broken, and the object is removed from the hierarchy. What happens if the deleted object is not a leaf node, but has descendants of its own? It is necessary to perform the same process for all of the object's descendants. Amputating the entire branch from the hierarchy, leads to some delicate considerations.

There is no way to guarantee that the subject deleting an object controls all of the descendants affected by such an operation. The operation as it stands can allow an object to have its access permissions set to null by a subject who may not actually control it, which seems to bypass the discretionary access mechanisms of the model. This is not truly the case, but it does imply that there exists a different interpretation of control for the Delete Object operation than that used for the Grant operation.

An undesirable possibility arises at this point. Consider the directory branch pictured in Figure 2, with two unclassified directories linked to two Top Secret directories. Assume that an unclassified subject has write access to the unclassified object at the top of the hierarchy. The access right allows the subject to control the object directly beneath the unclassified object.

Assume that, through clumsiness or malevolent intent, the subject decides to delete the object. The deletion is allowed to proceed, resulting in the removal of the object and the two Top Secret directories beneath it. The deletion of the lowest Top Secret object raises some questions. There is no way for an unclassified subject to control this object, since it could not get write access to the object's Top Secret ancestor. Nonetheless, the subject is able to delete the lowermost object which it does not control. This deletion is clearly undesirable, and implies that great care must be taken in the organization of directories. In addition, there are no mechanisms in the model which test to see if any of the objects to be deleted are currently accessed by other subjects. These are integrity and denial of service issues respectively, not secrecy. Since the original model incorporates a delete object rule within its purview, however, we point out the potential need for additional controls when using the model.

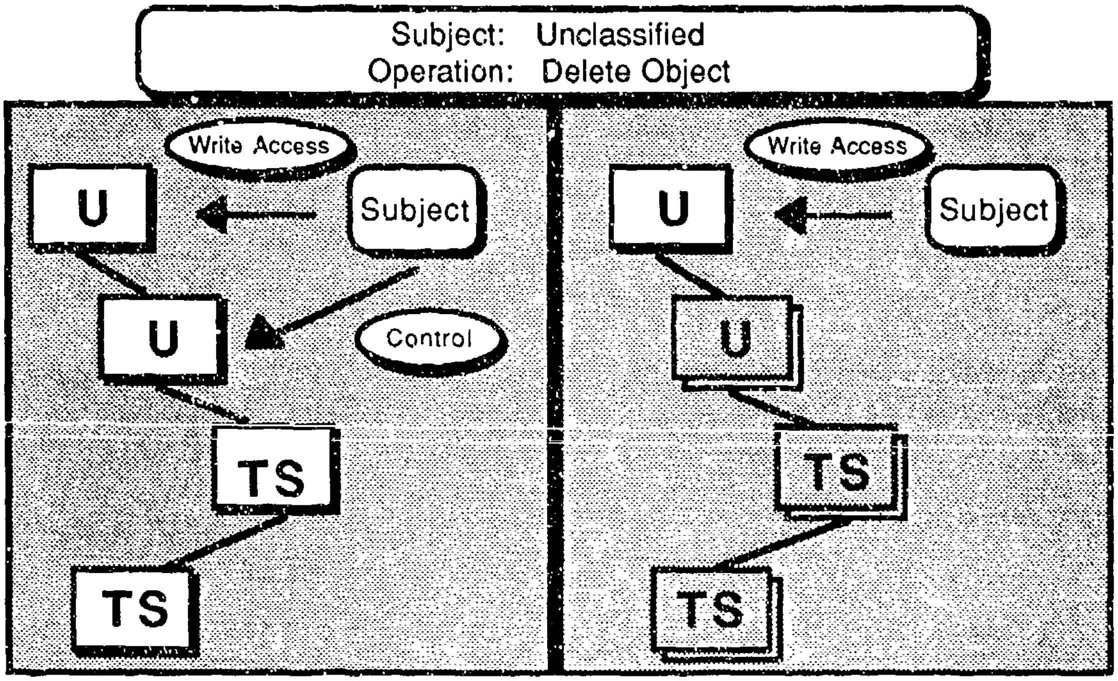


Figure 2. Delete Object Example

The Activity Principle

The BLP model is the only one of the three models studied which directly addressed the creation and deletion of objects. One of the difficulties initially encountered while generating rules for the Create Object and Delete Object operations was the absence of a formalized scheme of object reuse. An informal description of object reuse is found in the second volume of the model [5]. Unused objects are stored in an object pool of inactive objects. The creation of an object involves the activation of the index for an inactive object, while deletion corresponds to marking an active object inactive and returning it to the pool. The model assumes that only active objects may be accessed by subjects. This assumption is termed the Activity Principle.

What then is the status of an inactive object? The model text states that every active or inactive object has a security label.[4] (Its contents are assumed to have already been purged, in accordance with the Erasure Principle. However, since the model itself does not represent the contents of an object, there is no way to enforce this.) Bringing an object into creation requires

[4] See Reference 5, page 11: "Using this point of view, however, the created object O_j may have a classification and set of categories which do not match the requirements of the requesting subjects. (In the model, every object, active or not, has a classification and a set of categories assigned to it.)"

overwriting its original security label with the desired classification of the new object. Even though no data is currently associated with the object, overwriting an inactive object with a higher label with a lower one could be construed as a downgrade.

In order to generate effective rules for the Create Object and Delete Object operations, it is necessary to explicitly represent the Activity Principle for the model. Representation of the Activity Principle requires two primary additions to the model. The first is an `Active_Status` mapping for all objects which returns the active status of a given object. The second is a constraint which prohibits any subject from having access permissions to an inactive object which appears as:

```
dynamic constraint No_Access_To_Inactive_Objects is
begin
   for all sub: Subjects, ob : Objects |
  --- for all subjects and objects in the system it must be true that
   Active_Status(ob)! = inactive ->
  --- an object being inactive in the next state implies that
   access_permission(sub, ob) = null;
  --- each subject must have no access permissions to the object
end No_Access_To_Inactive_Objects;
```

Note that it was necessary to test the active status of the object in the *next* state. This test avoids the difficulties that arise when an object comes into existence *between* state transitions. Note that this constraint is enforced by limiting the access permissions to an inactive object, not current accesses. Since current accesses are always bound by discretionary permissions, this forbids any current accesses to an inactive object.

The rule base generated for this modified version of the model still contained some inconsistencies in the rule for the Create Object operation. These difficulties stemmed from the status of security labels for inactive objects. Overwriting the label of an inactive object while creating it could be interpreted as an uncontrolled downgrade, since the wording of the *-Property does not specify that it is only to be enforced over active objects. (None of the formalized properties of the model differentiate between active and inactive objects.) It was necessary to install a short-circuit clause of the form "**if** the object is active **then** no write down **else** the constraint does not apply" in the representation of the *-Property in order to achieve the proper results. While concerns such as these have long been addressed in the implementations based on a given model, the tools of the SMDE unearth these issues much earlier in the design process.

THE SRI MAC MODEL RULE BASE

Generating a rule base for the SRI SeaView Mandatory Access Control (MAC) model required more interpretation than the BLP model, because the model did not provide an explicit set of operations [6]. Since the accompanying text of the model suggested a likely set of primitives, it was not too difficult to define a set of operations. While the MAC model may seem loosely patterned after the BLP, it provided many novel features worthy of further investigation.

The Tranquility Principle

One of the salient characteristics of the MAC model is its reliance on state-independent functions to map subjects and objects to security labels. The use of state-independent functions was motivated by the need to facilitate theorem proving for the model. The result of this decision is a very strong interpretation of the Tranquility Principle for the MAC model. Since the Common Notation does not distinguish between state-dependent and state-independent mappings, it was necessary to find a means to enforce this interpretation of the Tranquility Principle within the confines of the Common Notation.

It was surprising to discover that this property was already enforced by default. Since none of the operations suggested for the MAC model attempt to change the security labels of subjects or objects, there is no immediate need to constrain the changing of labels. Therefore, we conclude that some model properties can be enforced by restricting the operations available to the model.

Nonetheless, we decided to test the ramifications of explicitly representing the tranquility of object labels by a constraint. This constraint, Object Tranquility, stipulates that the security labels of all objects must remain the same from state to state. In order to test the efficacy of this constraint, it was necessary to also define an operation which attempts to overwrite the label of an object with a new value. A new rule base was then generated for the model.

The rule generated for the new operation only allowed the operation to proceed if it overwrote the label with the same value. Although the tranquility of object labels is protected, an interesting question is posed. If this overwriting was somehow visible to the system, could it serve as a covert channel? Probably, yes, but this operation was not meant to be used so this is not a pressing problem, although it does highlight an interesting observation about current security models. Security models tend to constrain the values an object may take, but do not model the actual setting of a variable to a value. This allows the value of a variable to be overwritten with an identical value many times, as opposed to preventing the value from ever being reset.

Information Flow

The MAC model offers an innovative blend of information flow and access control concepts. While the model's access control mechanisms are quite similar to those of the BLP model, the information flow mechanisms represent a new point of departure. Generating a rule base which reflects the information flow concepts unearthed a few questions.

The MAC model addresses the flow of information via the Object_Contents mapping, which represents the data that objects contain. This feature is not found in the BLP model. This supports the modeling of the flow of information between objects. A write operation, as pictured in Figure 3, overwrites the contents of an object with a new value. If the contents of an object have changed during the course of an operation, it is said that information *flows* to the object.

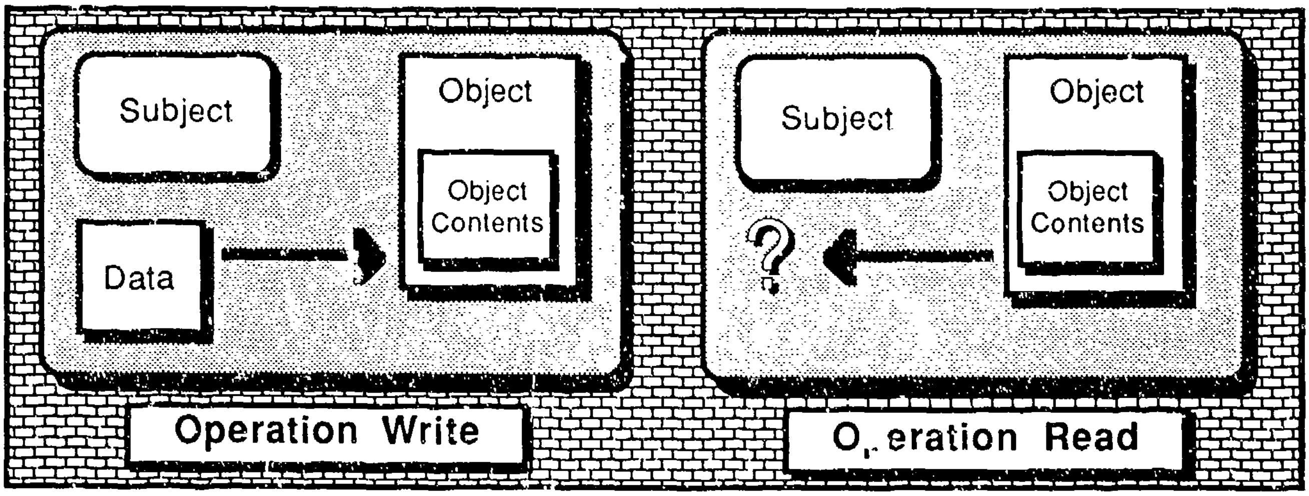


Figure 3. Write and Read Operations for the MAC Model

How is a read operation represented in the model? Intuitively, we can view it as a flow of information from the contents of the object to the subject. Unfortunately, the model does not provide a mechanism to store the information which flows to an object. Given the data structures of the model, a read operation creates no visible changes to the system state.

This issue is reflected in the model's Information Transition property which unifies the access control and information flow mechanisms of the model. This property states that, if the contents of an object are changed between states, then the subject initiating the operation must already have write access to the object. The property can be loosely paraphrased as "an operation that behaves like a write operation must have write access in order to execute." The effects of the Information Transition Property on read and write operations are portrayed in Figure 4.

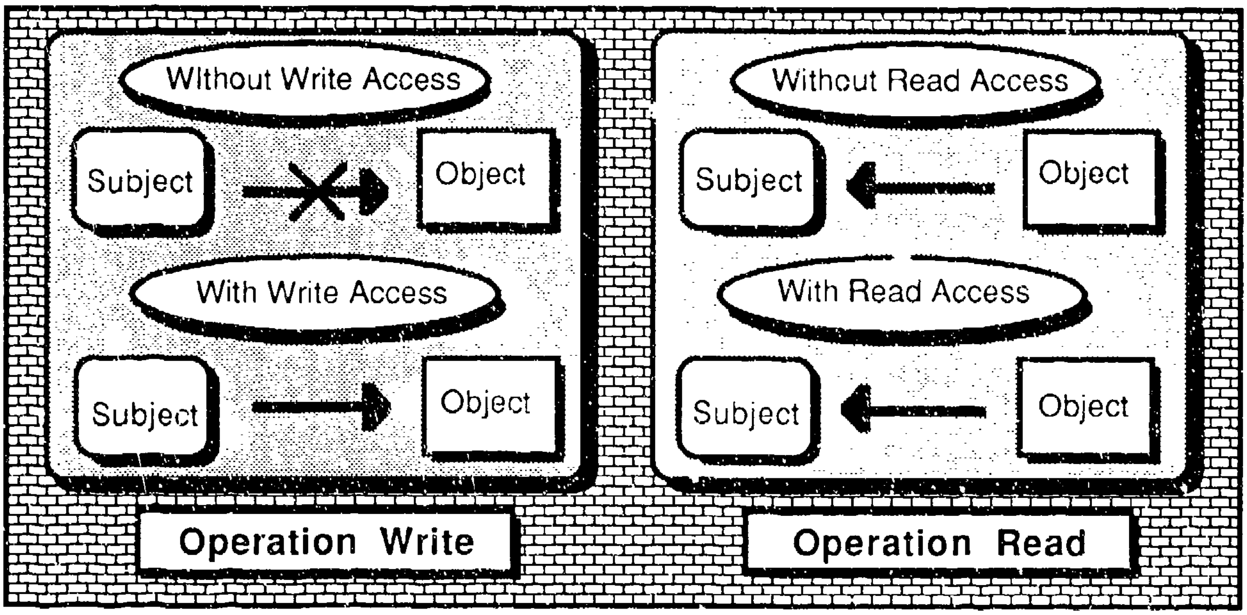


Figure 4. The Information Transition Property

What about the read operation? Does the Information Transition Property require a subject to have read access prior to reading the contents of an object? Since a read operation leaves no visible effect on the system state, it is impossible for the Information Transition Property to catch an unauthorized read operation. Although it is intuitively obvious that a subject would need read access prior to reading an object, this is not formalized in the model.

It should be noted that this issue has already been successfully addressed in the FTLS for the SeaView model [7]. The scheme employed took advantage of pre-operations and a Boolean flag to insure that no reads occur prior to obtaining read access. The authors of that report also commented that defining operations for the model proved to be invaluable in forcing intuitive assumptions out into the open.

Another solution to this question offers a more explicit modeling of information flow by challenging the established convention of using reads and writes as primitive operations. In a real system, these activities correspond to copying data between the I/O buffers representing objects and process buffers representing subjects. Would it be more natural to use a copy operation to represent reads and writes? Using copy would guarantee that the source and destination of the information flow are stated explicitly, allowing constraints on information flow to restrict the flow of information to subjects as well as objects. In the case of the MAC model, an additional Subject Contents mapping would be required to hold the data that flows to a subject during a read operation.

Virus Protection

How resistant is the MAC rule base to infection by a computer virus? A scenario was written to represent a viral attack, so that it could be simulated by the Testbed. The virus required some interpretation, since the model did not provide any mechanisms to represent the effects an execute operation would have on the system state. Write operations were used to simulate viral infection, since a virus would need to write to an executable file in order to infect it.

The rule base performed reasonably well in the face of viral attack. The Program-Integrity Property of the model insured that integrity and secrecy levels would not be compromised when a subject sought execute access to a given program. The secrecy controls alone were not very effective in warding off the virus, because secrecy controls are established to restrict the downward flow of information. Thus, an unclassified program could infect (write to) a Top Secret program, although the reverse would not be allowed. Since a virus is likely to infect a system through a commonly used utility program, secrecy controls would not prove very effective to counter it. The integrity controls were much more useful, since they aim to prevent the upward flow of corrupt information. They could protect sensitive programs from being infected by utilities. The Tranquility Principle also proved helpful in preventing the flow of a virus, since it prevented the virus from altering the label of an object to allow an infection where it would normally not take place.

THE MILITARY MESSAGE SYSTEM RULE BASE

The rule base derived for the Military Message System (MMS) model [8] reflects many of the innovations present in the original model. This model differs from many previous state machine models in that it places its emphasis on dynamic security properties. It also relies exclusively upon information flow controls to enforce mandatory security. Since the model was created to support a whole family of message systems, only one operation (Release) is formally defined. The Release operation models the application-specific task of formally releasing a message. Its inclusion in the model is mandated by the Release Secure property, which only allows the releaser field of an entity to be changed during the Release operation. (The Release Secure property is the only example we have seen to date of a security property explicitly referencing an operation.) The model also offers a powerful authorization mechanism with its use of entity access sets for discretionary access control. Creating a rule base for the model required defining a base set of operations. Defining operations for the model proved to be very helpful in understanding the model.

Entity Access Mechanisms

Discretionary access control in the MMS model is achieved by the use of access sets defined for each entity. The access set for a given entity is a set of triples of the form: (<User ID or role>, <operation>, <index), where <index> represents the position in the parameter list that the entity may legitimately occupy. The Access Secure property of the model insures that the operation may only proceed if, for all entities in the parameter list, there exists an entry in the access set which corresponds to the operation request. Consider the example in Figure 5. The third entry in the entity access set for `Entity 1` specifies that `User1` may use that entity in the `Copy Entity` operation when it occupies the second position in the parameter list. In this example, `User1` may copy from `Entity 1`, but not to `Entity 1`. This is roughly analogous to the statement that `User1` has "read permission" to `Entity 1`, but only in the context of the `Copy Entity` operation.

The entity access mechanisms provide a fine level of granularity for discretionary access control, since they make it possible to specify how each entity may be used by each user for any given operation. The fineness of granularity, however, encourages a much larger amount of overhead than would be required by the discretionary mechanisms of the BLP model. The task of defining accesses for a large number of operations and users for each entity would be time consuming as well.

Propagation of Access Permissions

The MMS model does not provide any notions of "ownership" such as that found in the BLP model. In the MMS model access permissions are modified through operations that allow a user to edit the entity access set. In our efforts it was necessary to define the operation `Update_AS`, which allows a user to specify the access set for a given entity. A user "controls" an entity if its access set authorizes the user to perform the `Update_AS` operation upon it. It is also conceivable that a user could extend `Update_AS` rights to another user, thereby extending control of the entity to someone else.

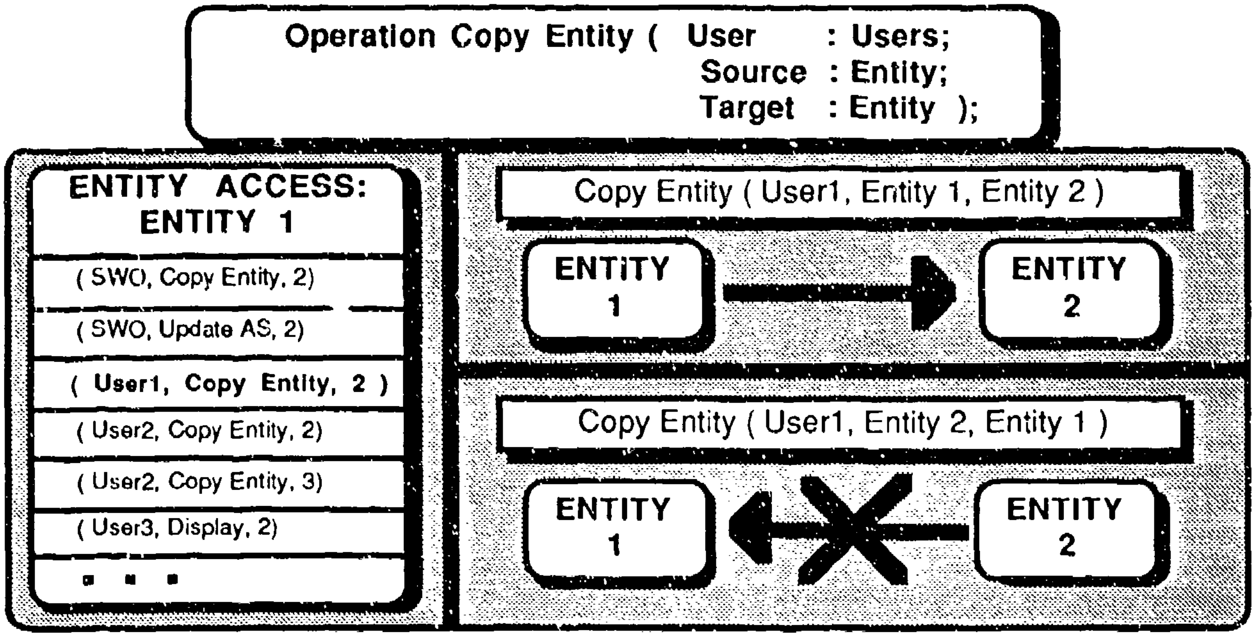


Figure 5. Entity Access

Can a malevolent user seize undisputed control of an entity once granted `Update_AS` rights? A scenario was written for the MMS rule base which duplicated the conditions described earlier for the BLP rule base. In this scenario, a benign user grants `Update_AS` permission to a malevolent user, who, in turn, attempts to seize control of the entity. Unlike the BLP rule base, the MMS rule base did not provide damage control mechanisms to limit the scope of abuse. Once granted `Update_AS` rights, the malevolent user was able to assume complete control of the entity. While it can be argued that an intelligent user would never extend `Update_AS` rights to another user in most situations, one must remember the large degree of overhead present in entity access sets. Consider a pressured environment where a user must make an entity available to another user. Faced with the prospect of defining every legitimate position the entity may take for every operation, the user may succumb to the temptation to extend `Update_AS` rights instead.

User Attributes

The user ID is a character string denoting a specific user. It is not classified as an entity per se, but as an extension of the user instead. One of the definitions underlying the model specifies that the mapping between users and user IDs be one-to-one, which prevents two users from sharing the same user ID. In order to test the strength of this property, an operation was defined, `Define_ID`, to overwrite the user ID for a given user with a new value. A new rule base was then derived for the model reflecting the addition of this operation.

The rule generated for `Define_ID` did contain the necessary check to insure that the one-to-one property of the ID mapping was preserved. However, it was surprising to note that there was no access set check incorporated by the rule, as would normally be required by the Access Secure property. Thus the user was allowed to alter his ID without explicit authorization. Why does this occur?

This unauthorized alteration results from the unique status of a user ID. Since it is viewed as an extension of the user and not as an entity, it does not have an entity access set to protect it from unauthorized use. Any operation being performed solely on user attributes does not need authorization, given a literal reading of the original model. Of course, the text accompanying the model [9] states that user attributes should only be modified by the System Security Officer (SSO). This stipulation could be formalized quite easily within the model. This example also touches on a theme introduced in the SRI MAC model discussion: constraints may be implicitly represented in the model by limiting the set of operations. If no operation is provided that would change a user ID, then there is no need to constrain the setting of user IDs. It would be possible to define a security model in terms of operations that would guarantee secure transitions to secure states. This approach would serve as the dual to the commonly accepted practice of basing models on security properties and leaving operations largely unspecified.

Copy Operations

Operations which copy data from one entity to another play a large role in the activity of a message system. Since the original model supports a family of systems, details concerning the implementation of copying are not specified. In creating a rule base for the MMS model, we defined three different copy operations, each of them altering the system state to a different degree. The first, `Copy Entity and Security Label`, copied the security label, releaser field and value from the source to the target entities. The second, `Copy Entity`, only transferred the value and releaser fields. This definitional difference was used to determine how the properties of the model would constrain any possible downgrade. The final copy operation, `Copy Value`, had the smallest effect on the system state. It merely copied the value from the source to the target. The different effects of each operation on the system state are illustrated in Figure 6.

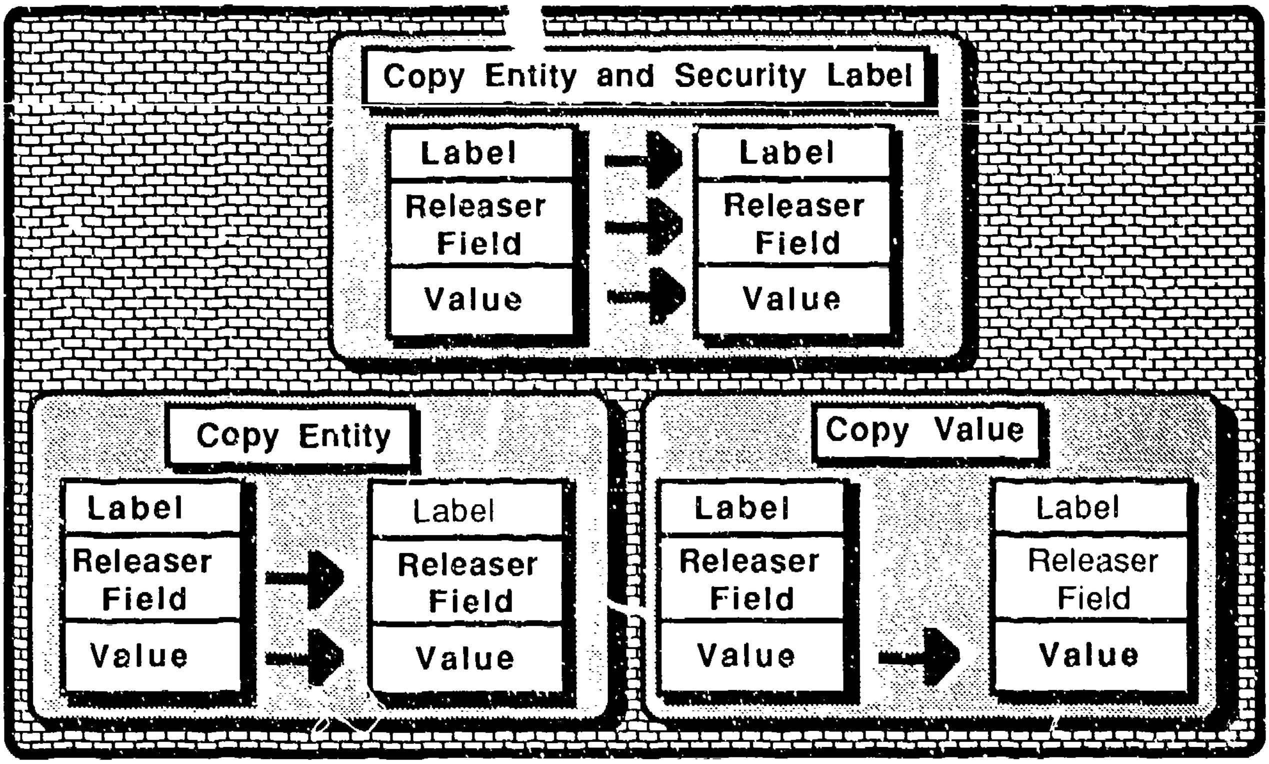


Figure 6. Three Copy Operations for the MMS Model

The rules generated for these three operations give a good insight into the behavior of the model's security properties. The rule generated for `Copy Entity and Security Label` did not include any clauses to prevent a downgrade because information extracted from the source inherited the proper classification. Since the `Copy Entity` operation did not copy the security label, its corresponding rule disallowed any downgrades unless the user had the downgrader role in their role set. The rules for both operations disallowed copying onto released messages when this would alter the target's releaser field. (The releaser field of an entity may only be modified by the Release operation.) The rule for the Copy Value operation, however, allowed any user to overwrite the value of a released message providing they had authorization via the entity access set. This overwriting seems undesirable from an intuitive standpoint and, like object deletion in the BLP, would require additional integrity controls.

Virus Protection

A scenario was written to simulate a viral attack on the MMS rule base. The viral attack was primarily modeled in terms of copy operations, since the model did not provide any mechanisms to represent the unique characteristics of executable code. Each attempted infection by the virus was a copy operation launched under the authorization of an unwitting user.

The MMS model seemed to be less resistant to viral infestation than the MAC model. The major weakness was a lack of integrity features. While secrecy controls halt the downward flow of valuable information, they do little to halt the upward flow of corrupt data. In order to prevent infection of an important program file from a corrupt compiler, the compiler must be classified at a higher level than the program. Highly classified source files, paradoxicall}, are best protected from viruses by lowering their classification

The entity access controls *do* offer a means to resist viruses. Combined with an effective administrative policy, they can shut down many of the entry points a virus would take into the system. In an application-oriented environment such as a message system, very few users should be authorized to modify executable code. Therefore, a strong policy of limiting users from writing to executable files could prove very effective if properly enforced. The only foreseeable shortcoming with this approach is that, if the user running the infected program has Update_AS rights, then an intelligent virus could attempt to modify the access set in order to allow an infection to take place.

CONCLUSIONS

The process of creating rule bases with a tool such as the SMDE offers a new paradigm for the creation, refinement and evaluation of security models. The impartiality of the Common Notation and Rule Generator force an explicit representation of the intuitive assumptions underlying models. The security properties of a model are examined in light of the operations a model is to support. Not only does this highlight issues which are often postponed until implementation, but it offers a dual to the commonly accepted approach of enforcing security in a model by properties. Careful definition and redefinition of a model's operation set can be as helpful in preserving system security as the definition of security properties. The simulation of a model rule base in action can offer a modeler valuable feedback that cannot be obtained by static methods of model analysis, such as the efficiency of modeling mechanisms which would no doubt be represented in the implementation based on such a model.

REFERENCES

[1] J. Heaney et al., Final Report for the Security Model Development Environment, A023, McLean, VA: Planning Research Corporation, December 1988.

[2] J. Page et al., "Strategic Defense Initiative Battle Management C3 Technology Program: Security Model Evaluation Via KB TCB Prototype Tools," A011, McLean, VA: Planning Research Corporation, June 1988.

[3] J. Heaney et al., "Strategic Defense Initiative Battle Management C3 Technology Program: Upgraded Common Notation," A007, McLean, VA: Planning Research Corporation, March 1988.

[4] D. E. Bell, Secure Computer Systems: A Refinement of the Mathematical Model, ESD-TR-73-273, Vol. III, Bedford, MA: The MITRE Corporation, April 1974.

[5] D. E. Bell and J. LaPadula, Secure Computer Systems: A Mathematical Model, ESD-TR-73-278, Vol. II, Bedford, MA: The MITRE Corporation, November 1973.

[6] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley, "The SeaView Formal Security Policy Model," A003: Interim Report, Menlo Park, CA: SRI International and Monterey, CA: Gemini Computers, July 1987.

[7] T. F. Lunt, and R. A. Whitehurst, "The SeaView Formal Top Level Specifications," A004: Interim Technical Report, Menlo Park, CA: SRI International, February 1988.

[8] J. McLean, C. E. Landwehr, and C. L. Heitmeyer, "A Formal Statement of the MMS Security Model," in Proceedings of the 1984 IEEE Symposium on Security and Privacy, April 29-May 2, 1984, pp. 188-194.

[9] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, "A Security Model for Mililtary Message Systems," NRL Report 8806, Washington, D.C.: Naval Research Laboratory, 1983.

HOOK-UP SECURITY
AND GENERALIZED RESTRICTIVENESS

Prof. Robert S. Lubarsky
Dept. of Mathematics, Franklin and Marshall College
PO Box 3003, Lancaster, PA 17604-3003 (717) 291-3872

topic: evaluation and certification

This work was supported by the Air Force Systems Command at Rome Air Development Center under contract #F30602-85-C-0056. The views and conclusions contained in this paper are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Air Force or the U.S. Government.

The author would like to thank Daryl McCullough for insight into the meaning of hook-up security, and Steve Vinter of BBN for many useful discussions on distributed system design. He would also like to acknowledge Doug Weber, whose help in the development and the writing of this paper would be hard to overestimate.

Much of the work contained herein was done while the author was employed by Odyssey Research Associates, Ithaca, NY.

1. Introduction

Recent work in computer security has centered around the notion of information not flowing in certain ways. For instance, there have been attempts to make precise the idea of information not flowing from one level to another and to verify this property of models of actual systems. A limitation of this approach is that in most real systems information does flow even between levels where it's not supposed to. This makes it impossible to prove that it doesn't.

There are some common examples of this phenomenon. One is that of downgrading. It is common that for the sake of flexibility a system will include a downgrading facility. The effects of this high-level act are clearly visible to a lower-level user, as they are supposed to be. There is also the case of limited access processes. Some system components can be accessed by only one user at a time, and will return a reject message if another tries to do so. So if a high-level user gets there first then this might be visible later to anyone. (Notice that this example is very similar to the leak caused by using the high water mark protocol.) Slightly different from these is the instance of uncertainty of the level of information. When someone tries to log on, it is unclear at first what the level of that message should be. There are any of a number of ways of formally labeling this message, but its real effect will be at the actual level of the user, which can be determined locally only after receiving the acknowledgement from the password database.

We would like to extend the current theory of security to handle such cases. Our approach is to determine a narrow range within which all the possible leaks occur, since once a user knows where and what the leaks are a different kind of analysis of them can help decide whether the system is acceptable regarding security. One reason to do this by generalizing what is already known is that some of the work would then be done for us. Yet this desire is not just pragmatic, it also follows from the ideas themselves. Consider restrictiveness (see below), the best current example of a security property. The intuition behind it is that all the information possibly available to a user at some security level l is unaffected by the inputs at levels not less than or equal to l. A crucial part of the formalization of this property is the restriction operator $\uparrow l$, which takes a sequence of messages and returns the subsequence of those messages at a level less than or equal to l. This is used to define the notion "everything that an l-user could possibly know". But if some high-level information does not remain strictly above l, then $\uparrow l$ is not the right restriction operator.

At this point one could attempt a simple generalization of $\uparrow l$. Instead of just throwing away a message with a high label, one could replace it with a message containing all of the information less than or equal to l. In the examples above, the message "downgrade X" would be replaced by "write(contents(X))", possibly with certain items, such as the identity of the user, also deleted. In a limited access process, the high-level command "I want you to do such-and-such" would be replaced by "Somebody wants to use you for something". For a login attempt, however it is actually labeled by the system, we would consider it at the level of the user, assuming that the users and their levels don't change.

Allowing for alterations of messages such as these, we could then define the view of a system to a user at level l, $\Uparrow l$, which takes a possible history of the system and returns what that history looks like at level l. $\Uparrow l$ can be defined inductively: $\langle\rangle \Uparrow l = \langle\rangle$, and $(\alpha{}^\wedge x) \Uparrow l = (\alpha \Uparrow l){}^\wedge m(\alpha, x)$, where $\langle\rangle$ is the empty string, ^ is concatenation, and m is some appropriate function. Presumably $m(\alpha,x) = x$ if the level of $x \leq l$, and is as suggested by the examples otherwise. Thus all the leaks are confined to those α and x such that $x \not\leq l$ but $m(\alpha, x) \neq \langle\rangle$, and they can be analyzed, possibly to get a quantitative measure of the rate of the leaks or to indicate to a supervisor where to check to see if leaks have occurred. Note that we allow the previous history as a parameter to m, as in the downgrading example.

Such an attempt, while mathematically sound, is in some measures inadequate. In the downgrading example, while the locus of information transfer is restricted to that one message, the content of the transfer is really unclear. On what does "contents(X)" depend? For limited access processes, presumably most of the calls to them do not interfere with one another, so by noting them all we carry around a lot of baggage which makes it seem as though more information is being transmitted than actually is. Regarding logins, we had to make the assumption that the users and their passwords are constant, which is related to the problem that the suggested function m cannot be computed locally.

All of these problems are related to the fact that we know what high-level information is available only retroactively. The downgraded message "write(contents(X))" should depend only on the writes to the file. We would like to retain those writes in $\alpha \Uparrow l$ and make m a function not of α and x but of $\alpha \Uparrow l$ and x. But any file might be downgraded, and saving the writes to all of them would defeat the purpose. m knows to retain writes to a downgraded file only retroactively. Similarly, the only holds on single-user processes of importance are those that later cause a reject message. Therefore m should retain the traces of only those requests, necessarily retroactively. For attempted logins, the situation is the clearest: the level of a login attempt is the level eventually assigned by the acknowledgement.

Another advantage of this more accurate modeling of real systems is that we are interested in not only what data somebody gets, but also when. As an example, when downgrading we would like to know not only that what the low-level user saw depended only on the writes to the file, but also that it didn't depend on even that much until a certain time.

As before, the restriction operator can be defined inductively, using the auxiliary function m. This time, though, whether to append $m(\alpha,x)$ or not may depend on later messages in the sequence. Also, $m(\alpha,x)$ depends upon $\alpha \Uparrow l$ instead of α, so we know that the information leaked is contained in what we have been saving.

What follows is a formalization of this latter approach. Examination of the details of this program reveals manipulations not found in the development of standard restrictiveness nor suggested by the intuitions above. We will try to explain and justify them as they occur. We begin with some basic definitions and notation, and an exposition of McCullough's restrictiveness. The centerpiece of the paper is the definition of $\Uparrow$, a generalized version of $\uparrow$, which allows for a generalization of restrictiveness.

2. Definitions and Notation

If X is a set, X^* is the set of all finite sequences of elements of X. If Y is a subset of X, $X \backslash Y$ is the set of objects in X and not in Y.

If $\alpha, \beta \in X^*$, $\alpha\hat{}\beta$ (the concatenation of α and β) is the sequence consisting of α followed by β. For $x \in X$, the sequence $\langle x \rangle$ is often identified with x itself.

$\alpha \uparrow Y$, the restriction of α with respect to Y, is the subsequence of α obtained by removing all components not in Y. Inductively, $\langle\rangle \uparrow Y = \langle\rangle$, $(\alpha\hat{}x) \uparrow Y = (\alpha \uparrow Y)\hat{}x$ if $x \in Y$, and $(\alpha\hat{}x) \uparrow Y = \alpha \uparrow Y$ if $x \notin Y$.

A process $P = \langle E, I, O, T \rangle$ is a set of events E, with disjoint subsets I and O of input and output events respectively, and $T \subseteq E^*$ the set of traces, thought of as all possible histories of the process. For the sake of this last property, all initial subsequences of a trace are traces: if $\alpha\hat{}\beta \in T$, then $\alpha \in T$.

A process with security is a process P with a function "level" from E to a partial order L of security levels. For example, L might be {unclassified, classified, secret, top secret}, ordered the obvious way, and level(e) is the sensitivity of the event e. If L contains the levels above for each of the incomparable categories Army, Navy, and NATO, then L is no longer a linear order, but a partial order. All processes will henceforth be assumed to be with security, even if not explicitly stated as such.

If l is a level (that is, $l \in L$), then $\alpha \uparrow l$ is $\alpha \uparrow \{x \mid \text{level}(x) \leq l\}$, and $\alpha \uparrow l^- = \alpha \uparrow \{x \mid \text{level}(x) \not\leq l\}$.

3. Hook-Up, Security, and Restrictiveness

Security is a difficult problem. One way to handle difficult problems is to break them up into smaller ones. After solving those, they must be pieced back together. It is the latter activity that concerns us now.

A computer system can be broken up into a collection of inter-communicating processes. To fit the pattern just suggested, we would like a security property such that, if it holds for each component, it holds for the whole system.

<u>def</u> Processes $P_1 = \langle E_1, I_1, O_1, T_1 \rangle$ and $P_2 = \langle E_2, I_2, O_2, T_2 \rangle$ are coherent if $E_1 \cap E_2 = (I_1 \cap O_2) \cup (I_2 \cap O_1)$: that is, their common events are those that are inputs to one and outputs to the other.

def If P_1 and P_2 are coherent, then their hook-up process $P_1 || P_2 = \langle E, I, O, T \rangle$ is such that

$$E = E_1 \cup E_2$$
$$I = I_1 \cup I_2 - (E_1 \cap E_2)$$
$$O = O_1 \cup O_2 - (E_1 \cap E_2)$$
$$t \in T \leftrightarrow (t \uparrow E_1) \in T_1 \text{ and } (t \uparrow E_2) \in T_2 .$$

def A property is a hook-up property if it holds of $P_1 || P_2$ whenever it holds of P_1 and of P_2 .

Regarding hook-up security properties, the first discovered as such seems to be restrictiveness, due to McCullough. The intuition behind it regards non-deducibility of higher-level inputs. It states that if some inputs invisible to a user at level l are changed, then there is a way of changing the future behavior of the system again in a manner invisible to the user.

def P is input-total if any trace may be extended by any input: $\forall \alpha \in T$ and $x \in I$, $\alpha \hat{} x \in T$.

def(McCullough) P is restrictive with respect to level l if it is input-total, and

$\forall \alpha, \gamma \in E^* \ \forall \beta, \beta' \in I^*$

if $\alpha \hat{} \beta \hat{} \gamma \in T$ and $\beta \uparrow l = \beta' \uparrow l$ then

$\exists \gamma' \in E^*$ such that $\alpha \hat{} \beta' \hat{} \gamma' \in T$,

$\gamma' \uparrow l = \gamma \uparrow l$, and $\gamma' \uparrow I \uparrow l^- = \langle\rangle$.

This notion is justified in part because it implies a more natural and apparently stronger non-deducibility property. (Notice that it does not preclude deducibility based on probabilities or on timing channels.) It is justified also by the following

theorem(McCullough) Restrictiveness is a hook-up property.

4. Generalized Restrictiveness

A. Limitable Processes and $\Uparrow$ A limitable process is a process P, along with a subset N of E and a function m: $E^* \times E \times E^* \rightarrow E$. These extra objects N and m are enough to allow us to define the restriction operator $\Uparrow$ described earlier, to allow for modeling limited information flow. To save on notation, we will drop the l from $\alpha \Uparrow l$, defining $\alpha \Uparrow$; the new notation is unambiguous since l is implicit in the choice of m . The empty sequence, $\langle\rangle$, is assumed to be an event. This way m can return $\langle\rangle$, allowing for

$$\alpha \wedge m(\beta, \epsilon, \gamma) = \alpha$$

As described above, $\Uparrow$ takes a sequence α and replaces each event with its low-level content (as given by m). What constitutes the low-level content of an event might be affected by future events, with $\alpha \Uparrow$ being merely the degenerate case of having no future events. So $\alpha \Uparrow$ is actually defined using an auxiliary notion $\alpha \Uparrow \beta$, where β is meant to be the sequence of events occurring after α . $\alpha \Uparrow \beta$ is defined inductively on α :

$$\langle\rangle \Uparrow \beta = \langle\rangle$$
$$\alpha{}^\wedge e \Uparrow \beta = \left[\alpha \Uparrow e{}^\wedge \beta\right]{}^\wedge m\left(\alpha \Uparrow e{}^\wedge \beta,\ e, \beta \uparrow E\backslash N\right).$$

Then $\alpha \Uparrow = \alpha \Uparrow \langle\rangle$.

Some explanation is in order. In general, m is the identity on some set S , such as the events at or beneath a given level. If m returns $\langle\rangle$ off of S , then $\Uparrow = \uparrow S$. Since we want to allow for some information to trickle through, we have m possibly extracting some information from an event e (in the context of the prior events α and future events β) . On what parameters should this extraction depend? Clearly it depends on m itself, which is assumed to be public knowledge. It should also depend on the previous history, or at least that part which is potentially visible, $\alpha \Uparrow e{}^\wedge \beta$, and also the current event e. It also must depend on future events, β, as described. But if we allow β as a parameter, we defeat the purpose of trying to pinpoint the influences upon $\Uparrow$. Using β as a parameter, we might permit highly classified information that it contains to trickle through. Therefore we select a presumably large body of events N to be the neutral events. They don't have the power to influence decisions about information flow. We focus all potential factors into the set $E\backslash N$ of non-neutral events. The effects that they could have are determined by m. So by studying $E\backslash N$ and m, we could understand the leaks, maybe even quantitatively.

Notice that we understand E to be sufficiently abstract. Sometimes m will return its middle argument cleared of much of its information, leaving something which could never be an actual message in a real system but which we consider an event. For instance, m might remove the client and the level from a downgrade message, leaving only that a certain file is to be downgraded. Such an event might never appear in any trace in T by virtue of its ungrammaticality, but we still consider it an event since we need it in the pseudo-histories $\alpha \Uparrow$

B. Examples The problems adduced as motivation were the login procedure, limited access processes, and downgrading. By way of illustrating this approach, we show how to express what is actually happening in these cases using our language.

To model the login, we consider a system with three components: a human user, the local host, and the login authenticator. The human's language includes the output "login request" at level X , the inputs "request approved" at each level l except X, and the input "request denied" at X. The host has all of those events with inputs and outputs reversed, along with the output "login check" at X, inputs "check approved" at each l except X, and the input "check denied" at X. The authenticator has the "check" events of the host, with inputs and outputs reversed.

A login attempt would consist of a request initiated by the human and passed along to the authenticator. This is at level X since so far no one outside of this small group can know anything about this sequence. The authenticator then consults its database, and either approves the login at a fixed level, or denies it again at an isolated level. This reply is then passed along to the human.

How would we define m to represent the view at level l? Requests and checks are invisible if they have not yet been confirmed, so $m(\alpha$, "login request or check", $\langle\rangle) = \langle\rangle$. Once the check is approved at level l, the check-event that caused it is visible at l: $m(\alpha$, "login check", "check approved at l") = "login check at l". Note that at this point "login check" is visible at l, while the "login request" that caused it is still at X, invisible to l. This is for reasons of coherence. That is, the host now knows enough to reclassify the request, but the human doesn't. So m_{host} could use the non-neutral event "check approved" to reclassify the request it received from the human, but m_{user} couldn't. To retain the coherence of the local m functions, the original request cannot yet be affected. The next event, though, is that the host transmits "request approved at l" to the human, and both processes reclassify the initiating request to l: $m(\alpha$, "login request", "request approved at l") = "login request at l". The neutral events are everything but the approvals. We speak more about the coherence of m's below.

For a limited access process, consider a file accessible to at most one user at a time. The languages for the clients each include outputs open, close, read, and write, and inputs confirmed and denied, at all levels. The language for the file is the same, with inputs and outputs reversed. The file will confirm an initial "open", then confirm any future sequence through the first "close", and wait to confirm the next "open". Anything else it denies.

When there is no leak, it suffices to use the standard restriction operator: $m(\alpha, x, \beta) = x$ if level$(x) \leq l$, $\langle\rangle$ otherwise. The only time there is a leak is when l tries to open the file and either it is currently being used by someone $\not\leq l$ or, following an earlier denial, the request is now confirmed. In the first case, the denial is tagged with an identifier for the currently operative "open". This is necessary so that in the inductive definition of $\Uparrow$ we know exactly which "open" to retain. So $m(\alpha, open(tag_0), denial\ (tag_1))$ = open if $tag_0 = tag_1$, $\langle\rangle$ otherwise, and $m(\alpha, denial(tag_1), \langle\rangle)$ = denial. Observe that m does strip off some information from $open(tag_0)$ and $denial(tag_1)$, since all that matters to the latest request is that somebody somewhere already has it. The second case is handled similarly, with the confirmation tagged with an identifier for the close that made it available. The neutral events here are everything except the denials and the subsequent confirmations.

For downgrading, the non-neutral message is "downgrade(X)". It makes visible the previous writes to X, removing all information such as client identities and levels from the writes and leaving only the content: $m(\alpha, write(tag), \langle\rangle) = \langle\rangle$; $m(\alpha, write(tag), downgrade)$ = write.

<u>C. Generalized Restrictiveness</u> By analogy with $\uparrow$ and restrictiveness, we say that a limitable process P is generalized restrictive (g.r.) if it is input-total and

$\forall \alpha, \alpha' \in E^*$ and $x \in N$, if

$$\alpha \Uparrow = \alpha' \Uparrow \text{ and}$$
$$\alpha\hat{\ }x,\ \alpha' \in T$$

then $\exists \beta' \in E^*$ and $\exists y \in E$ so that

$$\alpha\hat{\ }x \Uparrow = \alpha'\hat{\ }\beta'\hat{\ }y \Uparrow,\ \alpha'\hat{\ }\beta'\hat{\ }y \in T,$$
$$\beta' \uparrow I = \langle\rangle, \text{ and } \beta'\hat{\ }y \in N^*.$$

We can assume without loss of generality that $\alpha \Uparrow = \alpha'\hat{\ }\beta' \Uparrow$. (To see this, let β'' be the longest initial segment of β' such that $\alpha'\hat{\ }\beta''\Uparrow = \alpha\Uparrow$. Note that β'' might be $\langle\rangle$. Let y'' be the next event in $\beta'\hat{\ }y$ beyond β'' (which also may be $\langle\rangle$). Then β'' and y'' are as desired.

First we argue, necessarily informally, for why this is a useful property to use. Then we discuss its relationship to standard restrictiveness.

A sane notion of security is non-deducibility. A certain set of events w is secure from the view determined by m if:

for any trace α and legal sequence $u \in w^*$

there exists a trace β such that

$$\alpha \Uparrow = \beta \Uparrow \text{ and } \beta \uparrow w = u.$$

(A legal sequence is one which is realized by some trace: $u = \gamma \uparrow w$, for some $\gamma \in$ T.) With this property, an m-viewer can deduce nothing about $\alpha \uparrow w$. Usually the information we want secured are the inputs of level not less than or equal to l. In this context (letting $\Uparrow = \uparrow l$), these ideas are intuitive, precise, and their formalization is implied by restrictiveness.

In our more general setting such simplicity does not work. We might try to have w be those inputs with no l-effect. For starters we want more than that. If an input has an l-effect then it would not be in w, but if two have the same l-effect then we would not want to be able to distinguish between them. Even more seriously, "inputs with no l-effect" can not be well-defined, since m depends on the previous and future histories α and β as well as the current message x. Maybe sometimes an input is visible and other times not.

Our way to handle such problems, especially the second, is to consider deducibility of information in context, as a trace is being generated. The system is secured from deducibility if we cannot predict the future, nor find out that a previously reasonable guess as to the actual history was incorrect. This is meant to be necessary only when all the new events are neutral, so we can assume as much. That is, suppose that the real history α has been unfolding, and we have guessed that the actual history is α', based on our view l: $\alpha \Uparrow = \alpha' \Uparrow$. Then we are given the opportunity to guess those inputs with no l – effect, using only neutral events. Think of unrolling more of α' until all inputs before the next l – visible event occurs. In response, more of α is revealed, up to the next l – event, and including only neutral events. Note that we still have $\alpha \Uparrow = \alpha' \Uparrow$. Then the l – event x is revealed. Since it is also neutral, there is a way of extending α' to catch up with this new event. Without changing our earlier guess, nor our arbitrary prediction about future inputs, we can extend α' by neutral l – invisible non-inputs β', and then another neutral event y visible to l. The nature of y cannot be restricted beforehand, since m may be one – to – one, determining y completely. Still, in the general case we have circumscribed those events about which we can deduce something to those that are l – visible. Of course, given a particular m to analyze we can hope to do even better.

The assumption that all new events in sight are neutral is necessary. Suppose that α and α' are the same except that α includes a session in which a high-level user writes a file. If we extend α by the (non-neutral) x = "downgrade X" , that will affect the beginning part of $\alpha\text{^}x \Uparrow$. There's no way that the beginning part of α' can be so affected by any extension. If a non-neutral event is introduced, we may have to revise our earlier guess. It is for these instances that another kind of analysis in necessary. What we need is some measure of how much information comes through, possibly by measuring the changes forced upon α', when x is non-neutral. Then some judgement could be made about whether a particular system is acceptable for its purposes.

Generalized restrictiveness implies a limited form of standard restrictiveness. Using standard notation, to show standard restrictiveness, we are given certain α, β, γ, α', and β' ,and have to find a γ' with no inputs out of l. If γ has non-neutral events this may not be possible, so assume it doesn't. Consider the events of γ one by one. Use g.r. for each to find an appropriate extension with only neutral events, and at most one input out of l, that one being visible. So we can find a γ' , not with no non – l inputs, but whose only non – l inputs are l – visible, always avoiding $E \backslash N$. This is the best we could hope to do, given the set-up, and indeed it works.

D. Coherence and the Hook-up If the P_i are limitable processes, with associated functions m_i and sets N_i, they cohere if

- they cohere as processes,
- $N_0 \cap E_1 = N_1 \cap E_0$,
- for $\alpha, \beta \in E^*$ and x a common event

$m_0(\alpha_0, x, \beta_0) = m_1(\alpha_1, x, \beta_1)$

where $\alpha_i = \alpha \uparrow E_i$, and similarly for β_i, and

- if x is not in $E_{(1-i)}$ then $m_i\left(\alpha_i \Uparrow x \wedge \beta_i , x , \beta_i \uparrow E \backslash N\right)$ is also not in $E_{(1-i)}$.

If the P_i cohere as limitable processes, then $P = P_0 \,||\, P_1$ is also limitable, as follows. Let $N = N_0 \cup N_1$. By the second clause, we don't lose any non-neutral events. Let $m\colon E^* \times E \times E^* \to E$ be $m(\alpha, x, \beta) = m_i(\alpha_i, x, \beta_i)$, where x is in E_i . This is well-defined by the third requirement, and induces $\Uparrow : E^* \to E^*$. By the last, $\left(\alpha \Uparrow\right) \uparrow E_i = \left(\alpha \uparrow E_i\right) \Uparrow$.

Incidentally, the final clause is not just an technical convenience. It is necessary for security reasons. If $m_0(\alpha, x, \beta)$ is a low-level input from P_1, but $x \notin E_1$, then P_1 does not know to cover up for P_0's lie. This informal leak can be expressed formally.

E. The Hook-Up Theorem We would like to have the hook-up of two g.r. processes be g.r. This is not true, as the following example shows.

Let $E_0 = \{I', I_a, O_b\}$, $E_1 = \{I'', O_a, I_b\}$. $I_a = O_a, I_b = O_b$, and all other symbols are distinct. A symbol with an I is an input, an O is an output. Let $T_0 = I_0^* \cup (I' \wedge E_0^*)$; $T_1 = (I'' \wedge I_1^*) \cup (E_1 \backslash I'' \wedge E_1^*)$. Let $N_i = E_i$, and $m_i\left(\alpha, x, \beta\right) = \langle\rangle$ if $x = I'$ or I'' , I/O_b otherwise.

. It is easy to check that each P_i is a process (input-total, closed under initial segments, and disjoint inputs and output) and is g.r. Furthermore, the processes cohere and the m_i cohere. Nonetheless, $P_0 \,||\, P_1$ is not g.r. Let $\alpha = I'$, $\alpha' = I''$, and $x = O_b$. Notice that the aspect of retroactive changes is irrelevant here; even in the simpler case of replacing a message x by $m(x)$ we would have the same example.

The problem is that we need a certain amount of coordination between the processes. Each process agrees on what the restricted trace should look like, and can accommodate that with a real trace, but each insists that the real trace contain an input to itself. Neither is willing to put out.

Therefore, we say that a limitable process P puts out if, whenever $m\,(\alpha \Uparrow, x, \langle\rangle)$ is an output, $\alpha \wedge x \in T$ and $x \in N$, there exists β and there exists y so that $\alpha \wedge \beta \wedge y \in T$, $\alpha \wedge x \Uparrow = \alpha \wedge \beta \wedge y \Uparrow$, $\beta \in (N \backslash I)^*$, and y is a neutral output.

Theorem: If P_0 and P_1 cohere as limitable processes, and each is g.r. and puts out, then $P_0 \parallel P_1$ is g.r and puts out.

Sketch of proof: First we show g.r.

Given $\alpha\ \alpha'$ and x, we must find appropriate β' and y. If x is not in $E_{(1\text{-}i)}$ then apply g.r. to P_i only. This produces β'_i and y_i. Let $\beta' = \beta'_i$ and $y = y_i$. The important points to note are that $P_{(1\text{-}i)}$ finds this acceptable because all of the shared messages in β'_i are inputs to $P_{(1\text{-}i)}$ which is input-total. Also, $\Uparrow$ is unaffected on α' because β'_i contains only neutral elements.

If x is a shared event, let i be such that $m\ (\alpha \Uparrow, x, \langle\rangle)$ is an output for P_i. Use g.r. on $P_{(1\text{-}i)}$ to get $\beta'_{(1\text{-}i)}$ and $y_{(1\text{-}i)}$. Extend α' by $\beta'_{(1\text{-}i)}$. Any new common event is an input to P_i. Now apply g.r. for P_i to α_i and α'_i followed by the new inputs, $\beta'_{(1\text{-}i)} \uparrow I_i$. The later is a trace by input totality, and has the same view as α_i by the coherence o. m_0 and m_1. Extend $\alpha' \wedge \beta'_{(1\text{-}i)}$ by β'_i . If y_i is an output, extend again by y_i . If not, use the putting-out property. This yields β''_i and $y^{\wedge}$, which can be so appended.

The putting-out property is even easier to check. If $m\ (\alpha \Uparrow, x, \langle\rangle)$ is an output, then x is not a shared event. Therefore one can apply putting-out to the P_i such that x is in E_i. Notice that this uses only input-totality and not generalized restrictiveness in full, so that being input-total and putting out is itself a hook-up property.

References

[McCullough 87] McCullough, D. "Specifications for Multi-Level Security and a Hook-Up Property", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, May 1987.

[SDOS 88] BBN Laboratories, "The Secure Distributed Operating Design Project", RADC-TR 88-127, June 1988.

[Weber 87] Weber, D.G. and Lubarsky, R., "The SDOS Project -- Verifying Hook-up Security", *Proceedings of the 3rd Aerospace Computer Security Conference*, December 1987.

THE ARGUS SECURITY MODEL

Marc M. Adkins[1], Gary Dolsen, Jody Heaney, and John Page
Planning Research Corporation
Government Information Systems
1500 Planning Research Drive
McLean, Virginia 22102

Abstract

An overview of a new security model is presented. The Argus model features the use of copy instead of read and write as the lowest level information movement operation, and the combination of the Simple Security Property and the Security *-Property into a single information flow restriction. Other features include support for the handling of removable media (including hardcopy) and protection against viruses.

Introduction

The Argus computer security model described in this paper contains features of interest to the computer security community. While parts of the model are common to other current security models, these parts have been combined into a hybrid framework. On top of this we have implemented several new policies to extend the capabilities of the model to address areas not covered by other models.

The Argus model features the use of copy instead of read and write as the lowest level information movement operation, and the combination of the Simple Security Property and the Security *-Property into a single information flow restriction. The security perimeter has been extended to include the handling of objects on removable media (including hardcopy). Finally, the model includes special mechanisms for resisting viral infection and controlling the damage that may be done by Trojan horses.

This presentation is designed to provide an overall view of these features for critical review within the community. In addition, it may be desirable to incorporate aspects of the Argus model into other ongoing model development efforts. Due to space limitations, a complete explication of the model can not be provided. Full documentation of the model is contained in [1].

Motivation

The development of this model was part of a contract sponsored by the SDIO and monitored by Rome Air Development Center (RADC).[2] This particular sub-task was to use the tools developed under the contract to construct a hybrid model from relevant sections of other, existing models. The model was to be specified in the Common Notation[3] for model expression developed under the contract.

The Argus model combines features of the Bell and LaPadula (BLP), Military Message System (MMS), and SRI SeaView models. These models were evaluated during previous phases of the contract. The Argus model is a simple, general model with wide applicability, in the tradition of the BLP model. Unlike the BLP model, however, it is based on restricting information flows and utilizes a multi-level entity scheme, following the spirit of the MMS model. In addition, the Argus model incorporates two new policies that rely on the structure of the core model. This approach of

[1] Mr. Adkins is currently employed by Booz•Allen & Hamilton, Inc., 4330 East West Highway, Bethesda, Maryland 20814

[2] Contract number F30602 86-C-0190.

[3] The Common Notation is a language for expressing security models with surface similarity to the Ada® programming language. Developed for use with a set of automated tools, the Common Notation was used to express three existing models during earlier phases of the contract. Full documentation of the Common Notation is available in [2]. Ada® is a trademark of the United States Government (Ada Joint Program Office).

layered policies borrows a fundamental concept from the SRI SeaView model, wherein the application specific TCB model is built on top of the MAC model.

In the construction of the Argus model there were two main goals. The first was to address features of the above three models that appeared overly restrictive or incompletely specified. While it is not our intention to criticize particular models, previous work led us to some reservations about specific features (or their lack). The second goal was to add new features to address issues outside of the basic framework.

Existing Security Model Issues

In some existing models, the use of *read* and *write* as atomic operations does not provide enough information to check for a security violation. There are cases where the source and/or destination of an operation is implicit, and therefore untraceable. This is due to the way that the system state is represented. During a write, it is generally obvious that the object written to is changed, and this change is measurable either empirically or theoretically. During a read, however, some models do not represent the destination for the data read, and the source object does not change in any measurable manner. In other words, we can measure a difference between the new value of the object to which we have written and its old value, but not between the object from which we have read and *its* old value. Thus the read operation makes no observable change to the system state in these models (for a complete description of this problem see [3]).

Most existing models are not intended to handle the current trend towards multi-window workstation environments. For example, non-secure window-based environments allow users to cut and paste information between windows and interact in a variety of other "user-friendly" ways. Users of secure versions of these systems desire the same kind of interactivity, but the security-related questions raised by these operations are not handled well by models that treat devices as simple data receptacles.

In some models the file structure for the system to be implemented is specified explicitly in the model, making these models incompatible with certain tasks, functionality, or other models. For instance, the BLP model specifies a hierarchical file structure. In addition to providing a structure for the storage of files, the control attribute for a file is incorporated implicitly into the file structure. Any user with write permission to the ancestor of an object can alter the access permissions of that object. Thus, not only is the file structure specified in the model, aspects of the BLP model are dependent on its file structure.

New Issues

In addition to known model characteristics, we attempted to address some issues outside of the scope of models that we had studied. One such issue was the treatment of computer peripherals using removable media such as printers and disk drives. In non-secure computing facilities it is common to locate a high-speed printer in the computer center. Print jobs are sent to this printer and then picked up later. In a secure computing facility it is important to regulate access to the listings to provide some mechanism for assigning responsibility for the output of the printer.

Similarly, the handling and storage of removable media are the responsibility of computer center personnel operating on behalf of the entire user community. Currently this type of protection is implemented via physical security measures outside of the computer system. In general, connection between the security model of the system and these physical security measures is loosely specified at best. We theorized that a model could be built that extended the security perimeter to link up with the manual handling of computer media of various types.

It was also hypothesized that a type of virus protection could be built into a model by the special treatment of executable files. A computer virus is a code fragment hidden in another program. When that program is executed, the code fragment executes (without the user's knowledge) and creates a copy of itself in other program files. There may also be other side-effects, either benign or destructive. Generally the virus also acts as a Trojan horse once embedded within a host file, often with a time delay so that the damage is done at a later time (known as a "time bomb"). Since a virus breeds by spawning into executable files, it seemed that special protection for these files would be a first step towards preventing viral infection.

Argus Model Features

In the interests of brevity, a formal description of the Argus model in the Common Notation is not provided. Instead, the important aspects of the model are discussed in an informal manner. Crucial details are illustrated with fragments of Common Notation source code. The notation used should be, for the most part, self-evident. A full presentation of the model may be found in [1]. The Common Notation is described in [2].

The architecture of the Argus model consists of a number of support modules (referred to as *packages*) and the model itself, as shown in Exhibit 1. A number of lower level modules common to a variety of models are referred to as Support Packages. The Core Definitions of the model are data definitions and simple security restrictions that support those definitions. Basic Policy is provided in the form of the Mandatory and Discretionary packages. The Enhancements to the model embody the two additional features added to this basic framework.

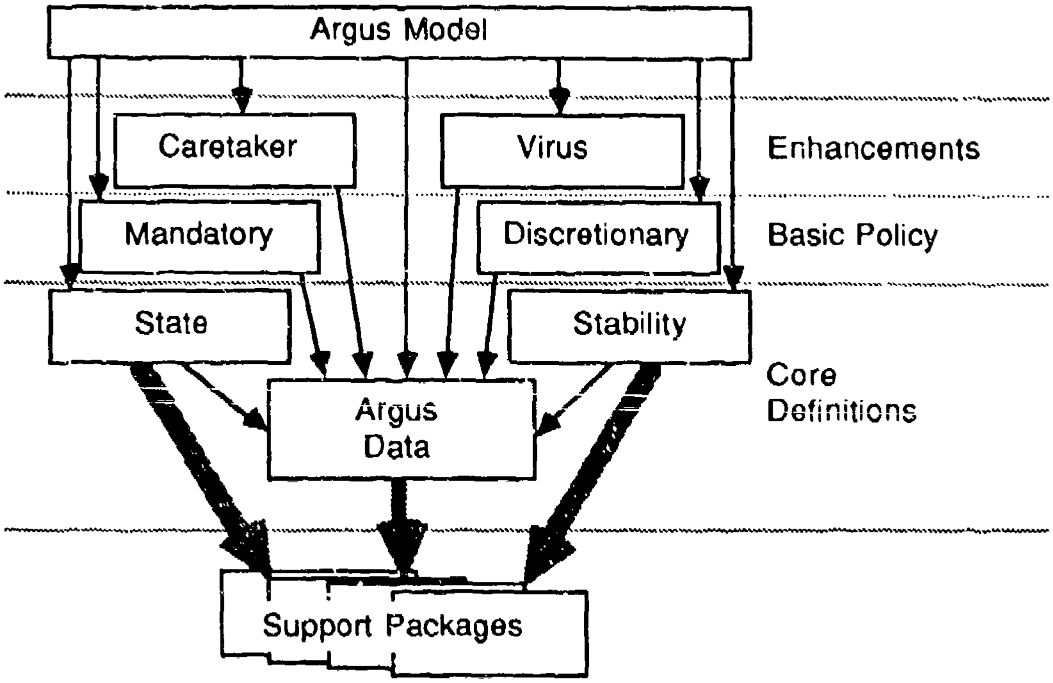


Exhibit 1. Argus Model Architecture.
Arrow denotes dependenc\ of a higher-level module on a lower one.

An important point about this architecture is that most of the packages are independent, allowing them to be removed or replaced with different functionality. For instance, the Caretaker package might be removed as unnecessary for a particular system implementation, without affecting the rest of the model. The Discretionary module might be replaced with one that supports usei roles in the tradition of the MMS model. This gives the Argus model a great deal of flexibility.

Data Structures

The basic entities of the Argus model are blocks, objects (files, processes, and displays), users, and devices. Security labels for some of these entities are specified as the accepted combination of level and category set. A security label *range* used for other entities consists of a low security label and a high security label. The data structures themselves are described in the Argus Data package. Restrictions on these data structures are contained in the State and Stability packages.

The most important data structures used in the Argus model are blocks, which represent single-level data entities, and objects, which represent multi-level data entities. Unlike the hierarchical data structures of the BLP and MMS models, the Argus model only allows a single level of nesting of blocks into objects. Exhibit 2 shows a typical object. On the left is a set of blocks containing the data that comprises the object. Each block has a security label, and the labels of the blocks are restricted to the security label *range* associated with the entire object. The set of data blocks for an object is considered the *value* of that object. Changes to the data blocks of the object are interpreted as changes to the object itself.

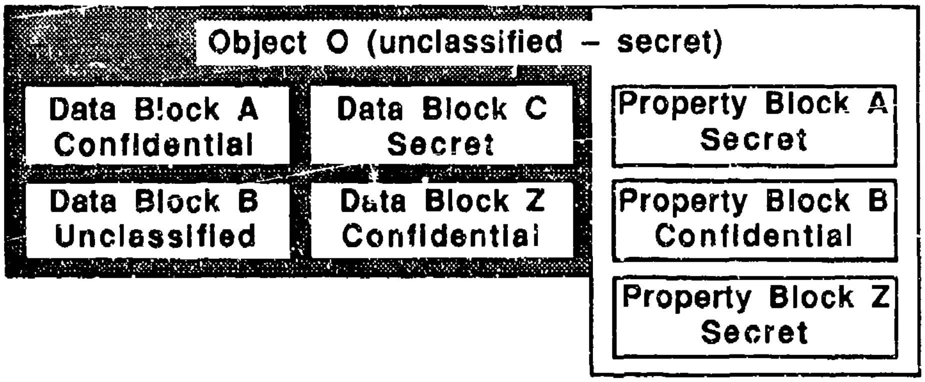

Object O (unclassified – secret)
Data Block A Confidential
Data Block C Secret
Data Block B Unclassified
Data Block Z Confidential
Property Block A Secret
Property Block B Confidential
Property Block Z Secret

Exhibit 2. Object and Block Structure within the Model.
Data blocks are on the left and property blocks on the right.
The security label range for the object is at the top.
Individual block security labels are inside of each block.

In addition to data blocks, each object may have *property* blocks describing aspects of the object that are outside of the actual data content of the object (shown at the right of Exhibit 2). These could include such things as file name, process priority, creation date, print form type, or file structure information. Since the properties are also blocks, they are protected by single level security labels which must conform to the security label range of the object itself.

The property mechanism allows file structure to be represented in a controlled but implementation-dependent manner. Note that the nesting of blocks into objects does not replace a file structure (or equivalent): each object would be one item in such a structure. Properties could be used to point to descendent and/or ancestor objects in a hierarchical structure, or this data might be kept within special directory objects. Since all properties are blocks, with their own security labels, this information is protected from users without proper clearance or need-to know. A particular example is the name of a file. If represented as a property, the file name may be set to a high security level, making it impossible to copy the property block to a display or process buffer of an improper security level. Thus objects may be made invisible to users with lower clearances.

There are three types of objects: files, processes, and displays. The *file* is a standard secondary storage repository for data. Each *process* is an execution session, representing a user. A process has a stack of current images, where an image is the executing form of an executable file. This stack represents the execution of programs by the process. A *display* is a document resident on an input/output device. Examples of displays include a page of hardcopy or a window on a video terminal. Since all objects are made up of blocks, these three types of objects are conceptually constructed from the same atomic units (blocks).

Users are the human users of the computer system. Each user has a security clearance, represented by a security label. This is the highest level of access that the user may exercise. Some users have special attributes: a Security Watch Officer (SWO) is a user with a wide range of special capabilities, and a downgrader is a user able to change the security label of data in a downward direction. The security label and special attributes of a user may only be changed by a SWO.

Each user owns the objects within the system that s/he has created. The security label of the owner must always dominate the high end of the security label range of the owned object. The security label range of an object may only be changed by the owner of that object. The owner of an object may only be changed by a SWO. In the case of processes, the owner of the process is considered the "current user" during operations of the process. This is true even for background o batch jobs (thus these jobs can not be run "by the system").

Devices are used to represent the hardware of the system. Objects are hosted on devices: files on secondary storage devices, processes on CPU devices, and displays on input/output devices. Objects do not migrate between hosts, an object is permanently resident on the host on which it is created (meaning that an object may only be moved between devices by copying it). Devices have security label ranges, and objects resident on a device must have security label ranges contained within that of the device. The security label range of a device may only be changed by the SWO.

The Copy Operation and Information Flow

The copy operation[4], used to copy the value of one block to another, is the most important operation of the Argus model for several reasons. First, the copying of blocks between different types of objects provides a common interface, supporting the movement of data between different types of media in a common, secure manner. Second, since objects cannot move between hosts, it is necessary to copy the blocks of an object from one host to another in order to "move" the object. Finally, it is the operation underlying both *read* or *write*, avoiding the problem of invisible read operations. The definition of the copy operation is contained in the model itself [1].

A copy represents the movement of data between any two blocks, regardless of the type of the objects to which the blocks belong or the devices on which these objects are hosted. While this may be implemented in a number of different ways in a particular implementation, the security relevant issue is always the movement of information from one block to another.

Instead of breaking data movement into separate read and write components, all data movement within the Argus model is described in terms of the copy operation. The basic copy operation is from one block to another (since blocks are always at a single security level, this is always a single-level copy). Using this model of computation, it is not necessary to have a read operation at the most basic level. The read is simply a copy from the source block (for example, a file record) to a destination block (for instance, a process I/O buffer). Likewise a write may be a copy from a process I/O buffer to a block on a printer display. By explicitly modeling the often ignored destination of a read operation, the Argus model is able to track the *entire* flow of data in any copy operation.

The use of *copy* as the basic operation is supported by the Common Notation **changes** operatoı and its implicit model of information flow. Since crucial aspects of the model are expressed using the **changes** operator, we digress briefly into a discussion of its semantics. In the Common Notation, an operation is specified as the effects that the execution of the operation will have on the system. For instance:

```
operation Copy (from_block, to_block: Blocks) is
  -- Copy data from one block, overwriting another block.
begin
  value_of(to_block) := value_of(from_block);
end Copy;
```

states that the data within the destination block will be overwritten by the data in the source block. Since this information is available, it is possible to express a clause such as:

```
block1 changes block2
```

[4] From this point on we use the term "operation" to refer to a transition of the system represented by the model from one state to another. This may be considered a function or operating system call, or a similar concept to the BLP "request." The security relevant aspects of an operation are those changes that are made to the entities that make up the system (the system state) during the execution of the operation.

within the body of a model constraint[5] and determine whether or not the clause is true. This determination is always done within the context of a particular operation. While this is simplistic, it does specify for any single operation the *potential* movement of information.

It should be noted here that the **changes** operator does not handle all classes of information flows. When discussing the information flows for an operation, we make the distinction between direct, indirect, transitive, and temporal information flows. A statement such as:

```
alpha := bravo;
```

in an operation represents a direct flow. Indirect flows are characterized by modification of the actual assignment statement by values other than the direct source. Examples:

```
(1)   alpha := array_var(index);
(2)   alpha := if var then value1 else value2 end if;
```

In example (1), an assignment from an array reference, the index of the reference (`index`) is the source of an indirect flow to `alpha`. In example (2), the variable `var` is the source of an indirect flow to `alpha`, since changing its value may be reflected in the value of `alpha`. In both cases the value of the indirect source has an effect on the value of the destination, but the source value *itself* is not necessarily transferred.

A transitive flow involves multiple assignments within a single operation. The example:

```
alpha := value1;
bravo := alpha;
```

demonstrates a transitive information flow from `value1` to `bravo` via the intermediate `alpha` (there is also a direct flow from `value1` to `alpha`). Note that these two statements must occur in the same operation to be a transitive flow.

A transitive-like flow that stretches across two or more separate operations is referred to as a temporal flow. Unlike the first three types of information flow, a temporal flow is not calculable except at run-time, since there is no way of determining (prior to run-time) the relative ordering of operations in a running system. The temporal flow is *not* covered by the **changes** operator, therefore user collusion and covert channels require separate analysis.

Mandatory and Discretionary Policies

These definitions are contained in the Mandatory and Discretionary packages.

The mandatory security policy is built directly upon the **changes** operator. This is done with a single constraint, subsuming both the Simple Security Property and the Security *-Property:

```
dynamic constraint Secure_Information_Flow is
-- Replaces both the Simple Security Property and the Security *-Property.
begin
  for all block1, block2 : Blocks |
    block1 changes block2 -> -- implies
      Dominates(security_label(block2),
                security_label(block1));
end Secure_Information_Flow;[6]
```

For information to flow from a source block to a destination block, the security label of the destination block must dominate the security label of the source block. Normally, the Simple Security property is used to block reading up and the Security *-Property is used to block writing down, but this constraint blocks both as they are just different ways of viewing the basic mechanism of copying

[5] We use the term *constraint* to refer to a security restriction with the model. An example of a constraint would be the Simple Security Property.

[6] This is similar to the Flow Policy from the SRI SeaView Policy document.

information. At the level of mandatory security control, the copy operation and **changes** operator allow an elegant replacement for both properties.

The mandatory policy is modified by the downgrader policy. In essence, the mandatory constraint may be circumvented if the owner of the process executing the operation (the "current user") is marked as a downgrader. In addition, a security label for a block may be changed in a downward direction only by a downgrader. In any event, the security label for a block must be within the security label range for the process executing the operation both before *and* after the operation. Thus a process may not reference a block outside of its label range, nor change a block label to be outside of its security label range.

The discretionary security policy is also built upon the concept of information flow between blocks, but in this case the user is presented with an interface based upon read and write access to multi-level objects. The actual restriction of information flow is best modeled using the **changes** operator, as with the mandatory control. However, actual access controls presented to the user must be of a more traditional type, as shown in the Discretionary_Information_Flow constraint:[7]

```
dynamic constraint Discretionary_Information_Flow is
-- Discretionary access to objects
begin
  for all block1, block2 : Blocks;
    block1 changes block2 -> -- implies
      read  in access_permissions    -- read from source legal
                  (owner(process), part_of(block1)) and
      write in access_permissions    -- write to destination legal
                  (owner(process), part_of(block2));
end Discretionary_Information_Flow;
```

where owner(process) refers to the current user and part_of(block) specifies the object of which block is a part.

The main difference between this formulation and the more traditional read/write based formulation is that the Discretionary_Information_Flow constraint is tracing the movement of information in a dynamic manner. The BLP mechanism, as a counter example, traces the start and end points of the copy, but does not respond to the *actual* read and write (not in any way modeled). In the Argus model, the copy operation is directly modeled, so the results of the operation may be seen in terms of the flow of information during that operation.

In order to ensure that the discretionary policy works correctly, it is necessary to constrain blocks to remain with a single object. To this end blocks are created already attached to a given object, and they do not move between objects. Blocks must be *copied* from one object to another (including the creation of a destination block if necessary). All blocks must belong to some (one) object at all times.

The ability to change the access permissions for an object is embodied in the control permission. When a user is granted control permission to an object, s/he is able to alter discretionary access permissions for that object. The owner of the object is constrained to always have control permission to that object (preventing hostile takeovers by other users with control permission).

Permission to execute an object is given with the execute permission. For an object to be executed by a process (present in the execution stack of that process) it is necessary that the owner of the process have execute permission to that object.

[7] For each user/object pair, a (possibly empty) set of access permissions is defined. A user has *access* permission to a given object if *access* is a member of the set defined for (user, object). We have defined the possible access permissions as read, write, control, and execute, where write is write-only and read/write access requires both read and write permissions.

Caretaker Protection

Physical devices are included in an extended security perimeter that makes users responsible for objects hosted on (or sent to) those devices. This mechanism is used to provide accountability for the physical handling of secure data that is produced by (or consumed by) the computer system. In particular, this policy is designed to handle the printing of documents on communal printers and the handling of removable secondary storage media.

Each physical device on the system is assigned a set of *caretakers*. The set of authorized caretakers for a device may only be changed by a SWO. At any given time, exactly one of these caretakers will have accepted responsibility for the device and its contents, which will be some set of data objects (as described above). The only time that there can be no caretaker for a device is when there are no objects on that device.

The responsible caretaker for a device can only be changed by a two-step process. First, the current caretaker informs the system of the identity of the next caretaker. Second, the new caretaker accepts responsibility for the device. This is modeled after the changing of the watch in a military environment. The two-step process prevents a caretaker from taking responsibility away from the currently responsible caretaker without his or her permission.

Constraint Caretaker_Label_Dominates places restrictions on what may be stored on a device:

```
constraint Caretaker_Label_Dominates is
begin
  for all object : Objects |
    -- Objects may not be placed on a device that does not have
    -- a responsible caretaker
    caretaker(host(object)) /= null
    and then
      -- The security labels of the objects on a device must be dominated by
      -- the clearance of the responsible caretaker for the device
      Dominates(security_label(caretaker(host(object))),
                label_range(object).high);
end Caretaker_Label_Dominates;
```

The combination of requiring a responsible caretaker for a device that hosts objects and requiring that the responsible caretaker's clearance dominate the classification of all hosted objects prevents the operator of the device (who would be the responsible caretaker) from access to data for which s/he is not cleared. In this manner, a communal device may be placed under the responsibility of a cleared operator who will be responsible for the distribution of the products of that device. This is patterned after the manual handling of hardcopy documents in a secure document control area.

The caretaker policy also extends to terminals used for access to the computer system. During the logon process, the user would accept responsibility for the terminal. This allows users to be restricted from using certain terminals (by removing the user from the authorized caretaker list for those terminals). Logoff would proceed by clearing all objects from the terminal and relinquishing responsibility to the system. During a session, the user would be responsible for all data displayed on the terminal.

This policy extends the security perimeter of the computer system to include the direct handling of media used by (or produced by) the system. Full accountability is preserved by the caretaker mechanism, providing a connection between the electronic and paper versions of the same data. At the same time, the mechanisms described here have been designed to mimic existing manual methods so that there should be no additional encumbrance on the security officers and computer operators involved in using such system based on the Argus model.

The Virus Policy

By representing the execution of programs and treating objects that contain executable images in a special manner, the Argus model reduces the entry window for viruses. This involves the identification of executable files, the identification of tools allowed to manipulate such files (if any), and a representation within the model of the execution of programs within a process. These three facets combine to reduce the chance of viral infection of a system.

Executable files, those that may be directly loaded and executed by the system, must be so marked by the system at the time of file creation. Only the SWO may change this attribute of a file (known as the image attribute) at any later time. A subset of these files (possibly empty) will qualify as linker files: this attribute is also protected such that it may only be changed by the SWO. Only linker files are allowed to manipulate image files.

The execution of a program loaded from an image file is referred to as an *image*. That is to say, a program is stored as an image file, but when loaded into a process it is referred to as an *image*. The representation of executing programs used in the model is a stack of images, with the currently executing image on the top of the stack. An image is considered to be suspended when another image is executing "on top of" it. Other models of execution may be desirable, but for the purpose of the virus mechanism it is only necessary to identify the image currently executing and the attributes of the file from whence it was loaded.

Viral protection is provided by restricting the modification of image files to processes currently executing images from linker files (as in Exhibit 3) by the Image_Manipulation_Security constraint:

```
dynamic constraint Image_Manipulation_Security is
--  Simplified version:
begin
  for all block1, block2 : Blocks |
  -- for any information transfer
    block1 changes block2
    -- where the destination block is part of an image file
    and then part_of(block2) : Files
    and then image(part_of(block2))
    -> -- the executing image must be a linker:
       linker(Top(execution_stack(process)));
end Image_Manipulation_Security;
```

Any virus (other than one that has been placed within a linker file) that attempts to modify another executable file will be prevented (as in Exhibit 4). Other than a collusion with the SWO, the only window for viruses is the linker itself, before it is placed on the system. After it is placed on the system it is protected because it is an image file. We intend that linkers be developed in secure environments and treated as highly classified data en route to computer sites, closing the loop around the system. Any system on which no development work is done needs no linkers, providing a virus-proof environment with no entry points.

Command files (or batch files) are not handled explicitly by the Argus model at this time. Since these are also subject to viruses, similar safeguards would be recommended. Unfortunately, most systems use simple text files as command files, making it difficult to provide the same type of controls. The solution appears to be some type of special-purpose command file editor that is marked similarly to the linker attribute and is the only legal tool for use with a specially marked command file. Similar techniques might be used in a development system to handle source files for various programming languages; or in a special-purpose system for data files connected with various applications.

In addition to virus protection, damage control for Trojan horses is provided by labelling the executable image objects as trusted or untrusted, and restricting the capabilities of untrusted images. This is in contrast to the Bell and LaPadula model (wherein a *process* would be trusted or untrusted) and serves a different purpose. In the Argus model, we intend that the trusted attribute for an image file be used to signify an executable image that is trusted to operate over multiple security levels simultaneously (the default for all newly created image files is to be untrusted). The trusted attribute is also protected so that it can only be changed by a SWO.

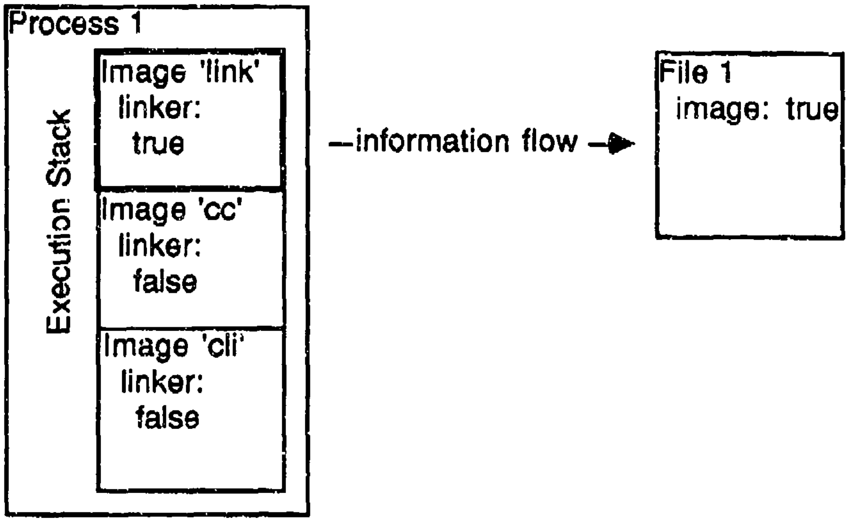


Exhibit 3. Legal manipulation of an image file by a linker.

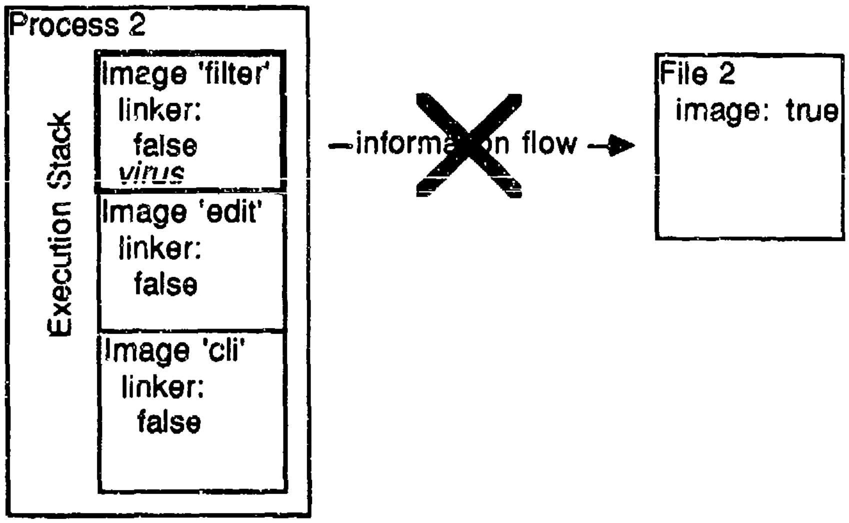

Process 2
Execution Stack
Image 'filter' linker: false virus
Image 'edit' linker: false
Image 'cli' linker: false
—information flow →
File 2 image: true

Exhibit 4. Attempted infection of an image file by a virus.

This does *not* bypass other constraints or policies, specifically the mandatory and downgrader policies. A trusted image is not allowed to downgrade information unless the normal downgrading requirements are met. Rather, a trusted image is allowed what would be considered normal execution in many other models. The security label range of an executable (`image`) file is used to restrict the range of security levels accessible by the executing image loaded from that file, but operations may be done within that entire range. For instance, a copy from a confidential block to a secret block is allowable for a trusted image marked confidential to secret.

By contrast, an untrusted image (from an executable file not marked with the `trusted` attribute) would only be able to use operations of a single level during its execution. This is modeled by restricting the security label range of a process to have its top label equal to its bottom label during the execution of an untrusted file. Thus, in the example above, an untrusted image from a file marked confidential to secret would be able to copy from confidential to confidential, but not from confidential to secret (or even from secret to secret in the same execution of the image).

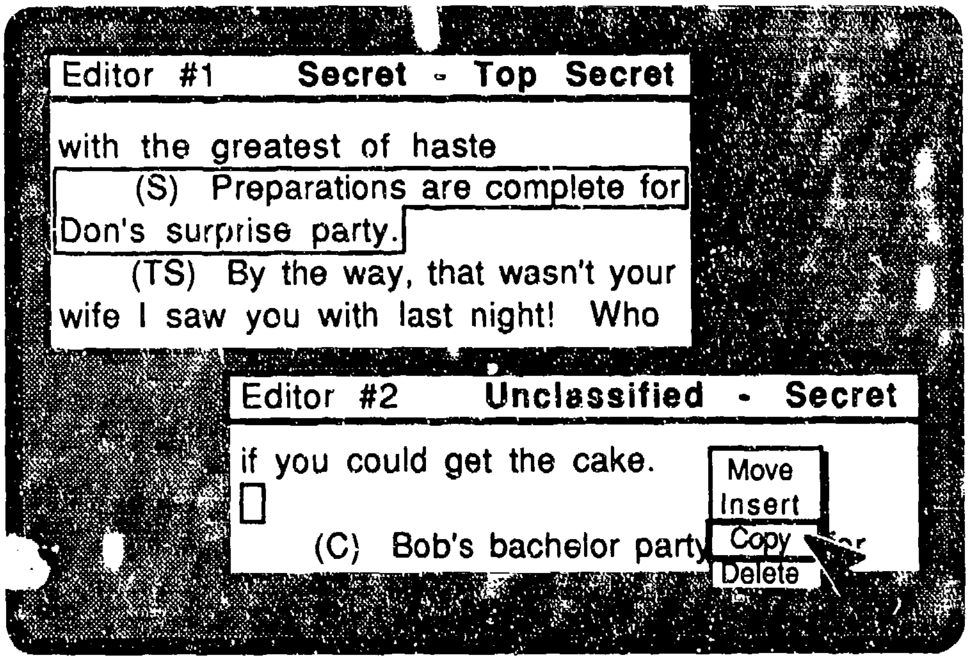
The intent of this is to prevent Trojan horses from rampant movement of data. In particular, an image with a Trojan horse should be prevented from downgrading data even if the current user is a downgrader. If such an image has not been explicitly marked `trusted`, then this cannot happen. (thus downgrading is only possible using `trusted image` files). It is up to the SWO to determine whether or not an image should be trusted in this sense.

Evaluation of the Argus Model

The model features presented in this paper demonstrate a number of interesting extensions to a hybrid model constructed by borrowing characteristics from several existing models. The basic framework of the model provides a flexible, multi-level object structure that is protected by information flow constraints. The combination of these features provides much more flexibility for implementors of both the system and utilities to be used on that system. Another major benefit is in the ability to provide a more flexible and productive (and less frustrating) user environment.

The use of *copy* as the atomic data manipulation operation explicitly specifies the source and destination of all data, thus providing enough information for security-relevant decisions to be made. In particular, this eliminates the use of implicit destinations for read operations, as the equivalent copy operation must have an explicit destination. Where necessary, the copy operation may be interpreted in terms of reads and writes (for instance the Discretionary policy), providing both mechanisms without conflict. Finally, the use of copy as the basic operation leads naturally to an information flow view of the entire model.

The block/object structure of the Argus model, coupled with the information flow mandatory security mechanism makes it possible to manipulate the contents of multi-level objects (such as workstation windows) in a straightforward manner. In particular, consider the case of two editing windows on a workstation, as illustrated in Exhibit 5. If the implementation of the editors takes advantage of the block structure of the displays (the windows) properly, the user will be able to copy secret data from one window to another (assuming that the security label ranges of the objects are appropriate) even though a mix of security levels is displayed in each window. This is the type of capability that makes a system more productive for the user, but is difficult to support with less flexible models.



Editor #1 Secret - Top Secret

with the greatest of haste

(S) Preparations are complete for Don's surprise party.

(TS) By the way, that wasn't your wife I saw you with last night! Who

Editor #2 Unclassified - Secret

if you could get the cake.

□

(C) Bob's bachelor party

Exhibit 5. Copying between windows on a workstation.

Using the property mechanism, implementors can construct arbitrarily complex, secure object structures (such as file structures) without there being a dependency within the model itself upon any such structure. In particular, the relational file structure proposed by Intermetrics for the Ada

Integrated Environment (AIE) in [4] and [5] would be possible within this model. This file structure allows any file to be marked as existing in a multi-dimensional space by assigning a property to the file for each dimension in which the file would be visible. Dimensions were intended to be attributes of the file, such as type of file (source, object, executable), version number of system into which file belongs, or author of the file. As mentioned above, these properties could be made secure to whatever level necessary, hiding entire dimensions of the file structure from uncleared users. Other models specifying dissimilar file structures would have no other way to handle the Intermetrics scheme, much less allow the user to add properties as necessary, whereas the property mechanism allows the file structure to be implementation specific *and* protected by the model.

The extension of the security perimeter to include human responsibility for physical devices provides a mechanism for handling hardcopy output from common devices and removable storage media in an auditable manner. Objects placed on devices are always under the responsibility of a known operator, providing positive control of removable media (both hardcopy and secondary storage). This linkage is modeled after manual data handling practices so that it will not present an unreasonable encumbrance, but will support the positive connection of system security to physical security.

The use of special file markings and explicit representation of the execution of images provides both a mechanism for reducing the entry window for viruses and limited damage control for Trojan horses. The entry window for viruses is reduced to a subset of executable files that is enabled (by the SWO) for writing to executable files. This subset may be subject to special physical protections to ensure that the correct file is entered onto the system and in some systems it may be empty, completely protecting the system from viral infection of executable files. While it is not possible to provide the same level of protection from Trojan horses, damage control is possible in a limited sense by only allowing trusted executable files to operate at multiple security levels simultaneously.

Summary

The Argus model demonstrates the construction of a hybrid model based on information flow concepts. This basic architecture supports flexibility on the part of the system and application developers. Ultimately, it is possible to provide the user with a more productive system.

The extensions to the Argus model attempt to address areas that are currently outside of the scope of the security model for a system. By bringing these issues into the security model, more assurance may be provided. Variations of these mechanisms may be useful in other models.

References

[1] Adkins, M. et al., Strategic Defense Initiative Battle Management C3 Technology Program: Hybrid Model Description, A014, Interim Technical Report, Planning Research Corporation, McLean, Virginia, 4 June 1988.

[2] Heaney, J. E. et al., Strategic Defense Initiative Battle Management C3 Technology Program: Upgraded Common Notation, A007, Interim Report, Planning Research Corporation, McLean, Virginia, 3 March 1988.

[3] Page, J. et al., Strategic Defense Initiative Battle Management C3 Technology Program: Security Model Evaluation Via KB TCB Prototype Tools, A011, Interim Technical Report, Planning Research Corporation, McLean, Virginia, 30 June 1988.

[4] Ada Integrated Environment (AIE) Design Rationale, Interim Technical Report, Intermetrics, Inc., Cambridge, Massachusetts, 13 March 1981

[5] Martin, Fred H., "Ada Integrated Environment, Executive Summary," Ada Update, Intermetrics, Inc., Cambridge, Massachusetts, December 1981

THE DESIGN OF THE TRUSTED WORKSTATION: A TRUE "INFOSEC" PRODUCT

Frank L. Mayer
J. Noelle McAuliffe

Trusted Information Systems, Inc.
3060 Washington Road (Route 97)
Glenwood, MD 21738

ABSTRACT

In recent years it has been recognized that the protection of classified and sensitive information in an distributed automated processing environment requires a total "information security" (INFOSEC) solution, combining both communications and computer security technologies into an integrated security solution. While the need for INFOSEC solutions is clearly recognized, the commercial availability of true INFOSEC products is extremely limited, or non-existent. This paper discusses the results of an effort to take commercially availsble COMSEC technology and commercially available trusted system technology, and integrate them into a readily available and evaluated INFOSEC product.

1. INTRODUCTION

The proliferation of commercially available products for the protection of sensitive and classified information is becoming a reality with the successes of the National Security Agency (NSA) Commercial COMSEC Endorsement Program (CCEP) and the National Computer Security Center (NCSC) program to evaluate commercially available trusted systems. In recent years, these security communities have come to realize that both arms of the *information security* (INFOSEC) problem, i.e., communication security (COMSEC) and computer security (COMPUSEC), are necessary to ensure the complete protection of sensitive information. However, the commercial availability of true INFOSEC products is nearly non-existent, despite the successes of the COMSEC and COMPUSEC halves of the problem. This paper discusses a product under development at Trusted Information Systems, Inc. (TIS), the Trusted Workstation (TWS), that is based upon commercially available COMSEC and trusted system products.

2. BACKGROUND

The major component of the COMSEC product, a personal computer encryption device (PCED), is an add-on board for an IBM PC compatible computer architecture. The PCED system also includes a plain/cipher switch, an interface for a key loader device, and an RS-232 asynchronous communications port (see Figure 1). The plain/cipher switch provides a user with the ability to determine whether data leaving the computer is encrypted. The PCED must be keyed with paper tape keying material via the key loader before being used for communication. The communication software approved for the PCED is written for the DOS operating system. The PCED is designed to allow users, utilizing ordinary personal computers, to communicate classified information over non-secure communication channels.

FTLS-BASED SECURITY TESTING FOR LOCK

Tad Taylor
Computational Logic, Inc.
3500 Westgate Drive - Suite #204
Durham, NC 27707

This work was performed by Computational Logic, Inc. for Honeywell Secure Computing Technology Center in support of National Computer Security Center contract MDA904-86-R-6544. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computation Logic, Inc., the Honeywell Secure Computing Technology Center, or the U.S. Government.

1. Introduction

The development of a thorough and convincing security test plan for an A1 system is a formidable task. It is made more difficult by the fact that there is little guidance available as to how to develop a set of test cases that result in increased assurance that a system is operating securely. A security testing approach based upon the FTLS (Formal Top Level Specification) was developed for the Honeywell LOCK (Logical Coprocessing Kernel) project as a possible method of addressing this issue. The approach is referred to as FTLS-based testing. This paper presents a high-level introduction to the concept of FTLS-based testing and how its use is being considered for the LOCK project[1].

FTLS-based testing is an innovative approach which pushes trusted system assurance beyond the A1 level of the Trusted Computer System Evaluation Criteria (TCSEC). The A1 level requires only an informal mapping of the FTLS to the source code. This results in a weak link between the FTLS and the implementation and is a frequently discussed deficiency of A1 assurance. A complete formal proof that the implementation (or at least the source code) is consistent with the FTLS would be much more convincing, but such a proof is still intractable for large secure system applications. FTLS-based testing is a compromise: it provides greater assurance of consistency than an informal mapping, but is substantially easier (i.e., more tractable) than an implementation proof.

We wish to point out that this approach has not been applied to any significant examples. It is only theory. Honeywell SCTC is studying the feasibility of using this approach. The objective of the study is to determine if this is indeed a viable option for LOCK, given constraints on such items as schedule, time, and money. However, it is hoped that the effort can be undertaken to serve as a validation of these ideas.

2. About LOCK

It is not necessary to understand how the LOCK system operates to understand the ideas put forth in this paper and it is beyond the scope of this paper to describe the LOCK system in any detail. The interested reader is referred to [Saydjari 87]. Briefly, LOCK is an approach to developing secure systems in which the vast majority of the security related processing capabilities of the system are physically isolated in a separate computer, known as the SIDEARM (System-Independent, Domain-Enforcing, Assured, Reference Monitor). LOCK technology is envisioned as being suitable for a wide variety of security applications. The current prototype effort calls for

[1] While the focus of this paper is on LOCK and the use of FTLS-based testing as a part of security testing, the approach should be equally applicable to helping establish safety criteria (other than security) about a system.

implementing a general purpose operating system on top of a LOCK base.

3. Formal Specification and Verification in the Development Process

In order to understand the role of FTLS-based testing for LOCK, it is necessary to understand the role of formal analysis in LOCK's development. Figure 1 represents a "traditional" software development waterfall chart, augmented by the inclusion of formal assurance techniques. At the top of the diagram, are a common set of requirements from which both the traditional and formal development efforts stem. For an A1 development effort, this would include the Trusted Computer System Evaluation Criteria (TCSEC) and a security policy.

The right side of Figure 1 (the traditional approach) consists of B-Specs, C-Specs, and finally, the implementation. The objective is to produce a running system that satisfies all requirements (performance, functionality, as well as security.) The LOCK B-Specs are a high level, procedural description of the system and are organized by functionality. They do not explicitly address overall operational constraints, such as security, but instead capture the functionality the system is intended to exhibit. This does not mean that security concerns are absent from the B-Specs; far from it. They include the functionality to be a secure system (e.g., checking access permissions), but they do not show that the specified functionality satisfies any given definition of security (e.g., does not define under what conditions access should be granted or denied). That is the role of other tasks. The C-Specs are a refinement of the B-Specs into a lower level system description. The implementation is produced by coding from the C-Specs.

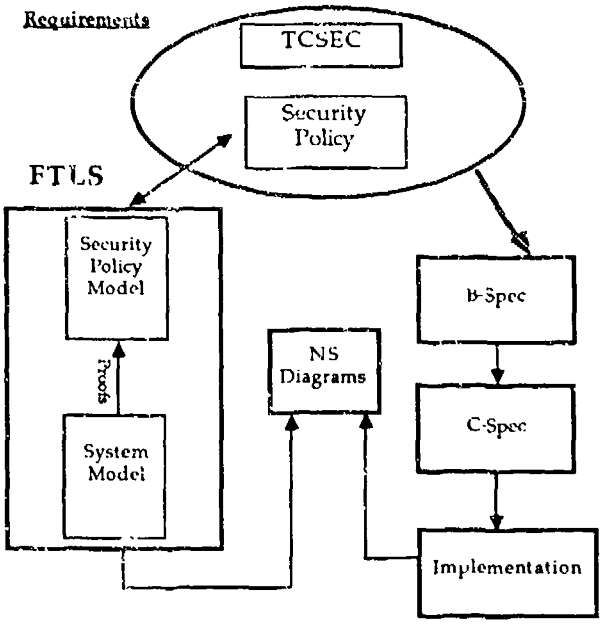


Figure 1: Waterfall Diagram of Software Development

The primary objective of the formal work (represented by the left side of Figure 1) is to provide increased assurance that the design of the system is secure. This objective is accomplished by formalizing the security policy, specifying the functionality of the system, and proving the functionality specified conforms to the security properties represented. The statement of the non-interference policy and the secure state invariants in the "Security Policy Model" capture what it means for LOCK to operate securely. The functionality of the system is captured by the System Model[2]. The System Model is intended to completely capture the user visible functionality of the TCB at

[2] The "conventional" view says that an FTLS is a formal non-procedural description of system behavior at an abstract level. In LOCK, the System Model portion of the FTLS serves this role.

an abstract level.

The arrow labeled "Proofs" represents the process of proving that the specified functionality is consistent with the constraints of the Security Policy Model, namely that all system effects are secure. The proof process is complex and involves utilizing "layers" of proof. Properties are proved for TCB requests. These "unit properties" are used to support proofs of system-wide properties, such as non-interference. This is explained more fully in section 4.

Finally, the two paradigms, formal and traditional, are explicitly ued together by showing a correspondence between the System Model and the implemented system. Correspondence, in this sense, means showing the implementation is consistent with the FTLS. This correspondence will be partially accomplished by mapping both the System Model and the implementation to the NS diagrams, as represented in Figure 1. NS (or Nassi-Shneiderman) diagrams are a flowcharting technique and are used as a common grounds of communication between the formal assurance group and the system developers.

While the traditional development paradigm views the system as a collection of interoperating components, the FTLS views system operation in a non-procedural, functional manner. The bulk of the B-Spec deals with describing the operation of the entities in the software architecture (i.e., how a particular task is accomplished.) However, the FTLS captures **what** is supposed to happen in response to a TCB request, but not how the actual implementation accomplishes that goal. Thus, entities strictly internal to the TCB may not be explicitly mentioned in the FTLS, only their effects that can be viewed from "outside" the TCB interface.

The difference between these two system views is presented in Figure 2. The "B-Spec View" describes the flow through the various system components, from client request to client response. The FTLS considers the relationship between client request and response (i.e., an input/output pairing) without dealing with the intermediate processing steps required to produce the result. The B-Spec view shows how a TCB request traps to the exception subject[3] which passes the request on to the host computer for processing and so on. Finally, a response is returned to the client and the system is in the resulting state. In the FTLS view, the resultant state and response are expressed strictly in terms of the input state. The intermediate stages are not considered. The end points of both views are intended to be equivalent. Establishing this equivalence is one of the purposes of FTLS-based testing.

4. Organization of the LOCK FTLS

Another important facet to understand concerning the LOCK project is the overall structure and style of the FTLS. It is an abstract, formal specification of the functionality evident at the TCB interface. It is a non-procedural specification, meaning that it describes **what** is the result of invoking a TCB request, but not **how** that effect is achieved. There is very little in the FTLS that specifies the mechanics of the LOCK system or any of the internal TCB structure. For example, the fact that the SIDEARM is physically separated from the host is not represented.

The FTLS is written in a state-machine style. The functionality of an individual TCB request is represented by a state transition function. For a given input state and set of applicable parameters, the function returns a new state reflecting any changes. These specifications are *definitional* in nature. That is, they exactly and completely describe the allowable effects on the system state of invoking a TCB request.

The FTLS consists of several parts, as represented by Figure 3. These are:

[3] The host entity to which clients issue TCB requests.

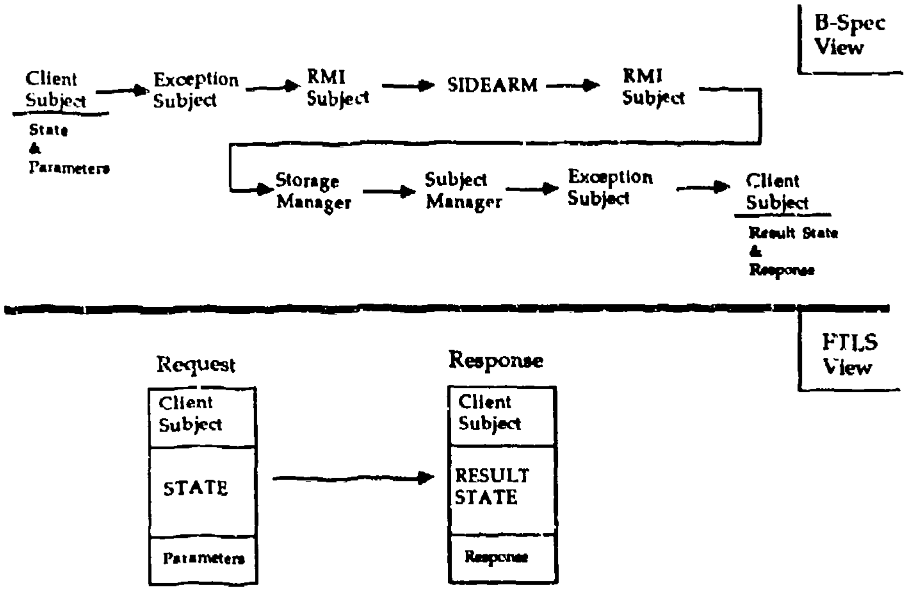


Figure 2: The Procedural B-Spec view of a TCB request compared to the FTLS non-procedural view.

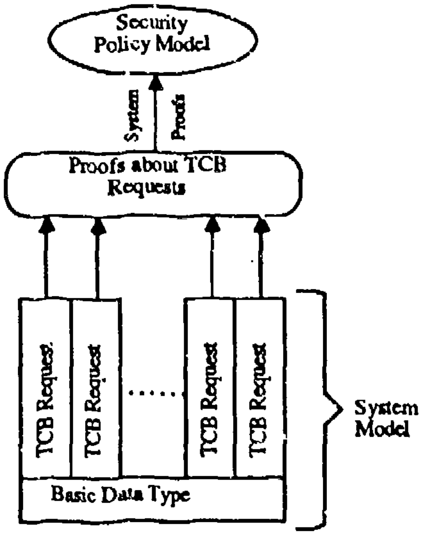


Figure 3: Structure of the Formal Top Level Specification

1. A statement of the LOCK security policy, labeled "security policy model."
2. An abstract description of the functionality of the TCB. Labeled "system model" in the diagram.
3. A set of proofs designed to show that the functionality specified satisfies the security policy. These proofs can, in turn be broken down into two groups:

   a. Proofs about individual TCB requests

   b. Proofs about the security of the whole system.

Proofs about individual TCB requests are those that show adherence to certain properties, such as maintaining state invariants. They are proved directly from the formal specification of the TCB request's functionality. Proofs about the system as a whole draw upon these proofs. By showing that all TCB requests have certain properties, it follows that the system as a whole has certain properties as well. It is this factoring of the total proof effort of which FTLS-based testing takes advantage.

5. The Concept of FTLS-Based Security Testing

The goal of FTLS-based security testing is to show that the operational semantics of the implemented system are secure, as defined by the Security Policy Model. This is achieved based on the fact that the essential result of the FTLS can be summarized as: "If the TCB functions as specified, then its operation is secure, as defined by the security policy model." It represents a rigorous and convincing argument that the design of the system is secure. By showing that the implementation is a valid instantiation of this specification, the arguments for security hold for it as well, at least to some extent. In many respects, this process could be considered a continuation of the formal development process, where a set of functional tests are used in the place of a set of proofs.

To develop effective security tests without using the LOCK FTLS, would require analyzing the security policy, factoring it into more primitive terms, and determining exactly what functionality could be exhibited by an individual TCB request and still maintain security. However, the LOCK FTLS has proved that the functionality specified for the TCB requests operates within the parameters defined by the security policy. Therefore, we have a description of allowable functionality to which we can test. By developing a set of functional tests to provide evidence that the implementation is operating according to the functionality specified in the FTLS, we gain additional confidence that the implemented system is secure.

$$\begin{tikzcd}
\text{Formal Specification} & x \arrow[r, "f"] & f(x) \\
\text{Implementation} & x_c \arrow[u] \arrow[r, "f_c"] & f_c(x_c) \arrow[u]
\end{tikzcd}$$

$$\boxed{\begin{array}{l} f_c \overset{map}{\approx} f \\ \& \\ x_c \overset{map}{\approx} x \\ \longrightarrow \\ f_c(x_c) \overset{map}{\approx} f(x) \end{array}}$$

Figure 4: Correspondence Between Specified Functionality and the Implementation

Figure 4 represents what must be shown in order to achieve this goal. A formal specification of an individual TCB request is represented by *f*. The specification represents a mapping from an input state[4], *x*, to a resulting state

[4] and any associated parameters

$f(x)$. The concrete realizations or implementation of x and f are represented by x_c and f_c, respectively.

The FTLS-based security tests are to establish the correspondence between the two views of system functionality, that of the FTLS and that of the implementation. More precisely, we want to show:

$$f_c \overset{map}{\approx} f \;\&\; x \overset{map}{\approx} x_c$$

$$\rightarrow$$

$$f(x) \overset{map}{\approx} f_c(x_c)$$

The symbol $\overset{map}{\approx}$ represents the mapping from an implementation state to an FTLS state. This expression means that if we are comparing the specification of a TCB request to its implementation and invoke both "machines" in equivalent states, then they should produce equivalent states.

The motivation for this approach is based upon the proofs associated with the FTLS. Several properties are proved about each f, based upon the specified functionality. The statement of these properties tends to be in the form $P(x) \rightarrow P(f(x))$, meaning that if some property, P, holds in a starting state, then P will hold in the state resulting from invoking f. From the results of a successful FTLS-based security testing effort, we would claim:

$$[f_c \overset{map}{\approx} f \;\&\; x \overset{map}{\approx} x_c$$

$$\rightarrow$$

$$f(x) \overset{map}{\approx} f_c(x_c)]$$

$$\&$$

$$[P(x) \rightarrow P(f(x))]$$

$$\rightarrow$$

$$[P(x_c) \rightarrow P(f_c(x_c))]$$

By this we mean, if the formal specification of a TCB request, f, has been shown to correspond with its implementation counterpart, f_c, AND it has been proved that f preserves property P, then we infer that the implementation preserves P as well.

6. Carrying Out the Approach in Practice

While the execution of this testing approach entails overcoming a certain amount of "gore", at a conceptual level, the process is straightforward. At this point, we wish to emphasize that this process has not, as yet, been carried out on examples of significant size. However, the process, as we envision it, is as follows:

1. A mapping between the state definition in the formal specification and the implementation, analogous to $\overset{map}{\approx}$ above, would be produced.
2. The formal specification would be partitioned into a set of separate conditions depending on the guard statements evident. This results in a set of "test schemas"[5]. These schemas are translated from the abstract representation of the Gypsy specification into the concrete terms of the implementation.
3. The test schemas are instantiated into a sufficient set of test cases and run on the implemented system. For each test case, the modifications to the implementation state are determined.
4. The Gypsy specification is used as an oracle for the testing process. This entails mapping the initial state information from the implementation (including any input data) to the abstract level of the formal specification, "executing" the specification and comparing the abstract and concrete final states.

[5] For want of a better word.

The following sections describe these steps in more detail.

6.1 State Mapping

For each state component in the FTLS, there is a well defined entity in the implementation that corresponds to it. Furthermore, every implementation state entity that influences the response to a TCB request should be represented by some aspect of the FTLS's security state, albeit at an abstract level. This correspondence is to be established by other methods, e.g., code inspection.

6.2 Partitioning the Testing State Space

Test schemas are defined for each TCB request based on the functionality described in the FTLS. In LOCK, the general form of a TCB formal specification is:

```
if <requirements for successful completion>
  then <modify state for successful effects>
  else if <condition defining particular irregularity>
  then <appropriate action>
  ......
  else <general failure>
```

Each logical condition defined in the request will lead to various test schemas. This partitioning ensures that the state space, as defined by the FTLS, receives complete coverage during the testing process. This is similar in concept to testing all control paths in an implementation, but, in general, decidably more tractable.

These test schemas define a set of test cases. They state that under certain conditions, certain effects should occur from invoking a TCB request. However, these statements use FTLS terms and definitions. It is the role of the next task to turn these schema into usable test cases.

6.3 Develop Concrete Test Cases

Once a set of test schemas are extracted from the FTLS, it is necessary to transform them into concrete instances of the conditions they define. The set of test data is defined based on the data types of the state and the conditions identified in the FTLS. The data types point out boundary conditions, minimum and maximum values, etc. The definitions of relevant data types and intuition play a role in this step. Obviously, it is crucial to define a sufficient set of concrete test cases to thoroughly exercise each of the FTLS's test schemas.

6.4 Running the Tests

Since, the FTLS specifies exactly what changes take place to the state as a result of invoking a TCB request, it is not enough to ensure that the explicitly specified changes occur when they are supposed to and don't when they aren't. It must be shown that nothing else changes. In the case of any TCB request, it is not sufficient to merely test that data structures and status flags receive the proper values under the right circumstances. It must also be shown that nothing else of "importance" changes in the implementation state. "Importance" is signified by corresponding to the FTLS state. It is this *a priori* determination of what's important and what's not that helps to make this security testing approach feasible. It allows us to determine which functions to trace and which values to check after a TCB request is invoked.

For every test case, those portions of the state that are not to be modified must remain inviolate. Therefore, in order to determine whether this condition is being met, it is necessary to determine state "deltas" resulting from running tests. In this sense, a state delta reflects all changes to the state, not just those that were anticipated.

6.5 Evaluating the Results

Evaluating the results of the various test cases is a matter of comparing them to the values predicted by the FTLS. This requires evaluating the FTLS for a given set of data. Information should be extracted at the start of each test execution to provide starting state information for the FTLS.

7. Benefits

We believe there are several benefits in utilizing the LOCK FTLS for the security testing effort. Some of these are pointed out below.

- Utilizes Best Statement of Security Policy. The FTLS contains a detailed statement of LOCK security. Obviously, such a definition is required for security testing. Furthermore, the FTLS has made the necessary transformations and interpretations between an abstract statement of security and the real effects of invoking a TCB request.
- Builds upon Related Efforts. A great deal of effort and analysis has gone into the development and verification of the LOCK FTLS. The overall LOCK security policy has been formalized and the functionality of the TCB units has been shown to be consistent with it. If the FTLS was not used for the security testing effort, exactly the same stages and interpretations would have to be made and it would be done with less rigor.
- Focuses on One Definition of Security. Many previous efforts to develop secure systems have been hindered by the fact that each "group" (i.e., the formal assurance group and the actual development group) tends to develop their own concept of security. Comparisons between these differing views of what it means to be "secure" are often confusing and unconvincing. However, this approach focuses on one definition of security, namely the one represented in the FTLS (the one given the utmost scrutiny.)
- Definitive Testing Results. The functional statements of the LOCK FTLS are clear and relatively concise. It is anticipated to be straightforward to determine whether or not a given test case has succeeded.
- Bounded Testing. The process is bounded by the relatively narrow focus and well-definedness of what is being established by the security testing process, i.e., functional equivalence, not abstract properties. Compare this to more open-ended forms of testing, such as penetration testing, or resource-bounded testing (i.e., testing until time and/or money is exhausted.)
- Criteria Requirements. There is a specific Criteria requirement to use security testing to establish a correspondence between the FTLS and the implementation. This effort would satisfy that requirement.

8. The Fine Print

In spite of the glowing statements made above, no claim is being made that an FTLS-based testing approach completely solves the problem of developing an effective security test plan for A1 systems. There are both limitations and risks associated with the approach, as well as assumptions that must be addressed.

8.1 Limitations

This approach does not completely address Security Testing. The successful completion of all the test sets defined will not constitute a complete set of security testing evidence, only one part of a larger picture. Other items that must be addressed include:

- Validation of the FTLS's assumptions.
- Validation of the FTLS's definition of the system state.

- Aspects of security not covered by the Security Policy Model represented in the FTLS.
- Covert Channel Analysis and related security testing.
- Penetration testing.
- Successful completion of other testing, such as unit testing.

Many of these other techniques are used to support the validity of the FTLS-based testing approach. For example, penetration testing can be an effective means of identifying additional functionality. Testing from any set of specifications would have difficulty in catching such a situation.

8.2 Risks

While there is much to be gained by this approach to Security Testing, it is not without some risk. The main risk seems to be the reliance being placed upon the FTLS. Any flaws in it will adversely affect the security testing process. The effects could range from causing delays in the testing process to hiding flaws in the design of the system. For example, if the Gypsy specification of a TCB request is wrong, the intended functionality would have to be determined, respecified and reproved. On the other hand, if the proof of system security was flawed in a way that allowed insecurities, then showing that the implementation exhibits the specified functionality would allow that same insecurity.

While these could represent severe impacts to the project, in all likelihood, the FTLS is probably the safest way to go. No other aspect of the development process receives such intense scrutiny from such a variety of sources. Therefore, serious mistakes seem least likely in the formal process. Not using the results of the formal analysis would mean dividing resources and duplication of effort. While redundant efforts might uncover more mistakes, it is a very expensive process.

8.3 Assumptions

There are several basic assumptions underlying the premise of this testing approach that must be addressed before the approach can be considered useful. These include the following points:

- The security policy model is accepted as valid.
- The proof strategy utilized by the FTLS is valid and the proofs are sound.
- The FTLS completely captures the security-relevant aspects of the implementation.
- The underlying assumptions made by the FTLS are valid. An example is that hardware mechanisms (e.g., an MMU) operate as anticipated.
- All potential interactions between subjects and the TCB interface are captured by the FTLS.

Some of these assumptions can be validated by other techniques, especially many of the FTLS assumptions. For example, a set of primitive tests could be developed to gain assurance that the MMU functions as believed and a form of code inspection could be applied to show that the FTLS was complete.

9. Spec/Code Correspondence and FTLS-Based Security Testing

The need for a mapping from the FTLS to the implementation is called for in the Criteria in section 4.1.4.4 where it states:

> The TCB implementation (i.e., in hardware, firmware, and software) shall be informally shown to be consistent with the **formal top-level specification(FTLS)**. The elements of the FTLS shall be shown, using informal techniques, to correspond to the elements of the TCB.

There exists a very close connection between the FTLS-based security testing approach defined above and the mapping from the formal specification to the implementation required for A1 systems. To start, both share the same goal, to show that the implementation is consistent or ''corresponds'' with the FTLS. In both cases, the underlying motivation for this requirement is to show that the formal analysis applied to the FTLS has some relevance with respect to the implemented system.

LOCK's current plans for spec/code include mapping both the FTLS and the implementation to a common set of Nassi-Shneiderman diagrams. While this goes further than most (if not all) previous such efforts, it still suffers from a lack of convincing evidence that the semantics of the FTLS and implementation are equivalent. For example, if the NS diagram said ''Determine if access is allowed,'' it is unclear that this will be interpreted in the same fashion in both the FTLS and the implementation. Something more is needed. The FTLS-based testing approach defined above could be exactly that ''something more.''

We believe that the results of an FTLS-based testing effort are sufficiently strong enough to support a spec/code effort so that it meets both the requirements and spirit of the TCSEC. Since a convincing spec/code requires a substantial amount of effort in its own right, this may lead to a cost savings of the total effort required.

10. Conclusions

This paper has presented an approach for developing test cases to support an A1 security testing effort based upon the FTLS. The approach could just as easily be applied to developing test cases for FTLSes that dealt with issues other than security or were not specifically targeted for A1-evaluation. FTLS-based security testing strives to provide evidence that a system is actually implemented in a secure manner. This is a much stronger result than conventional security testing methods could produce. We believe that it is one of the most promising techniques on the horizon for strengthening the A1 paradigm in a meaningful and cost-effective manner.

11. Acknowledgments

I would like to thank Honeywell SCTC for the opportunity to develop these ideas. In particular, Tom Haigh has provided insightful comments and has been very supportive. The comments of Bret Hartman, of Computational Logic, Inc., have been very valuable. Of course, any flaws or misstatements within this paper are the sole responsibility of the author.

References

[Saydjari 87] O.Sami Saydjari, Joseph M. Beckman, Jeffrey R. Leaman.
Locking Computers Securely.
In Proceeding of National Computer Security Conference, 1987.

FORMAL SPECIFICATION OF A SECURE DISTRIBUTED MESSAGING SYSTEM

Vijay Varadharajan Stewart Black

Hewlett-Packard Laboratories, Filton Road,
Stoke Gifford, Bristol BS12 6QZ, U.K.

June 22, 1989

Abstract

This paper describes the formal specification of security aspects of a messaging system architecture. The messaging system being considered is the X.400 Message Handling System (MHS) and the security architecture includes a number of security features described in the CCITT X.400 Recommendations. In this paper, we use the internationally standardised Formal Description Technique LOTOS (Language of Temporal Ordering Specification) for specifying the security aspects. We first describe the security aspects of the messaging system and then describe the modelling of these aspects in LOTOS. Finally we discuss how they relate to the overall messaging model, and draw some co clusions.

1. Introduction

Information Security is becoming increasingly important because of the growing need for "open" networked systems. It is being more and more recognised that security issues play an important part in the design of distributed systems and databases. In particular in applications such as electronic mail and electronic funds transfer, security is becoming an essential element of the services being offered. An area of considerable interest is the development of an open standard electronic mail service. In this paper, we consider the Message Handling System (MHS) as described in the CCITT X.400 Recommendations ([1]).

In this paper, we describe a suitable security architecture for the Message Handling System. The first step in developing a security architecture is to identify the constituent parts of the system and the likely security threats that can be mounted against them. Then the required security services and mechanisms needed to provide security can be defined. For instance, the OSI Security Architecture describes the type of security services and mechanisms that can be employed within the OSI Reference Model ([2]).

The security architecture described in this paper has been developed as part of the LOCATOR collaborative project, which is itself part of the Mobile Information Systems Project, a major demonstrator within the UK Government sponsored Alvey Programme. The partners within the LOCATOR Project are Hewlett-Packard Ltd., Racal-Milgo Ltd., Racal-Research Ltd., Racal Imaging Systems and University College London. The security architecture has been developed by the LOCATOR project team and they give a complete description of the security architecture in ([3]). The developed security architecture conforms to the CCITT Draft Recommendations. Here we briefly mention the relevant features of the security architecture which are required for our formal specification work. Currently, the LOCATOR Project is nearing the end of its implementation phase.

We *formally* specify these security services and mechanisms using the Formal Description Technique LOTOS ([5]). The formal specification of security aspects has proven to be very useful in many respects. It has allowed us to isolate and model only the security issues involved in the design of this system. It has enabled us to investigate the type of constructs and expressive power needed for modelling security. This process has also helped to make explicit some of the assumptions that have been made at the design level. Finally, the specification has provided the necessary abstraction allowing representation of architectural aspects and the hiding of implementation details.

The paper is intended to be as self-contained as possible and it is divided into the following parts. The first part gives a brief overview of the Message Handling System architecture (Section 2). In the second part we describe the security aspects of the messaging system and how it prevents the potential threats to the system. The third part describes the modelling of the security aspects of the system using the Formal Descriptive Technique LOTOS. We conclude by outlining some further security related problems in such a distributed system and assessing the suitability and usefulness of LOTOS for modelling security aspects. We also briefly discuss some tools to help this modelling process and to simulate the specification.

2. Message Handling System Architecture

In this Section we give a brief description of the general Message Handling System (MHS) used in the CCITT X.400 Recommendations ([1]). The entire MHS application is considered as occupying the Application layer of the OSI Reference Model. A Message Handling System is a set of computer processes which cooperate to provide the users with a reliable means of store and forward capability for their message transfer. To clarify the different needs during the Message Transfer, the MHS system has been divided into the following parts : User Agent, Message Transfer System, Message Transfer Agent and Message Store.

(a) User Agent (UA) : This is a process which interfaces with the "User" on one side, and with the "Message Transfer System" on the other side. The User Agent allows the user to create and submit messages to the Message Transfer System, and it collects messages from the Message Transfer System and presents them to the user.

(b) Message Transfer System (MTS) : The Message Transfer System is used to physically move the messages between computers and it consists of a number of cooperating Message Transfer Agents (MTAs).

(c) Message Transfer Agent (MTA) : A Message Transfer Agent is a computer that routes and relays messages. An MTA cooperates with other MTAs to relay and deliver messages to the appropriate User Agent.

(d) Message Store (MS) : The Message Store acts as an intermediary between the User Agents and the Message Transfer Agents and it provides reliable storage of delivered messages and thereby gives more control over the receipt of messages.

Figure 1 shows the basic components of the Message Handling System model. A message created by a user is submitted to the MTS via a UA. The MTA forwards this message towards the final delivery point within the MTS, which is the MTA attached to the UA whose address is given in the message.

A UA may either reside in the same computer as the MTA or it can be connected to an MTA by some network. In the first case, the UA accesses the MTS elements of service by interacting directly with the MTA. In the second case, the UA communicates with the MTA via the standard protocols. Several UAs may be attached to a single MTA.

An MS can be co-located with the UA , co-located with the MTA, or stand alone. We will briefly outline in Section 6 the effect of the inclusion of the Message Store on the security services provided in our architecture.

The protocols between these components are also shown in Figure 1. Protocol P1 is concerned with the transfer of messages between the MTAs and is called the Message Transfer Protocol. Protocol P3 is the MTS access protocol between the MTS and the MS. Protocol P7 is the MS access protocol between the MTA and the UA. The P2 protocol is between the UAs in the system.

2.1 Security Threats in the Messaging System

The distributed nature of the messaging system makes it susceptible to a number of security threats. Typical threats include eavesdropping and disclosure of information to unauthorized

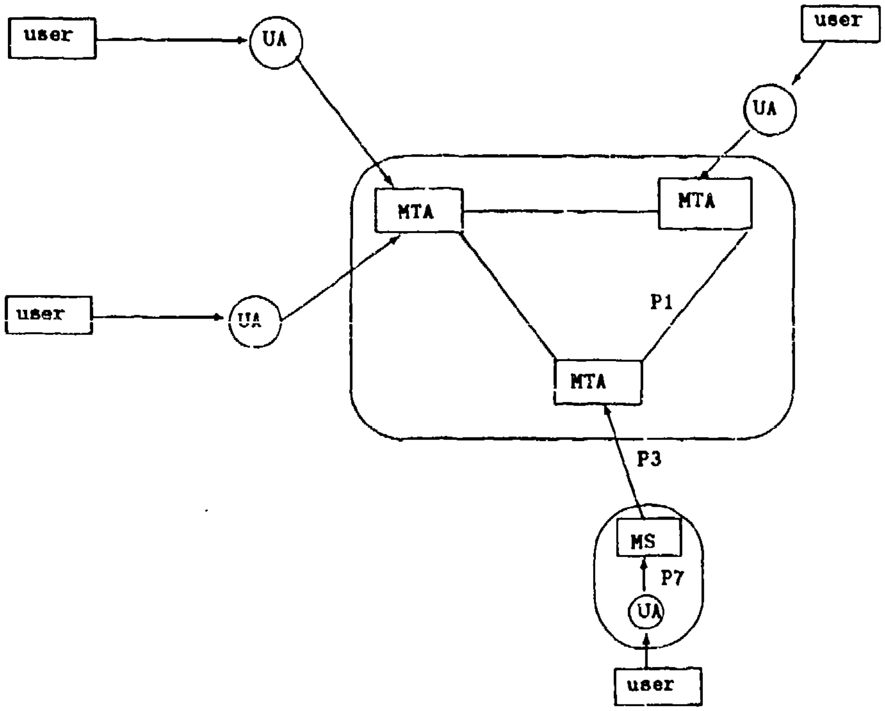


Figure 1: Functional Model of the Message Handling System

users, one user masquerading as another user, modification of messages while being transferred, and a user denying the sending or receiving of messages.

To overcome these threats several security services have been provided in the X.400 Recommendations. These security services include confidentiality, integrity, authentication and non-repudiation. We will be considering these security services and their formal specification in more detail in the later sections of this paper. It is important to mention that the application of these services requires the use of cryptographic keys, which in turn requires some sort of Key Distribution Centre to manage these keys. For this purpose, the Directory Service (as proposed in the X.500 Recommendations ([4])) has been used.

2.2 Directory Services

The Directory Service is required to support the security services within the messaging system and to provide a Name Server. Typically, the MHS may access the Directory to determine the credentials of a user for the authentication process, identify the intended receiver and to resolve the expansion of distribution list names. The two basic entities of the Directory Service are the Directory User Agent and the Directory Service Agent.

The Directory User Agent (DUA) : The DUA helps an entity to formulate and submit requests to the Directory and it also receives and formats the results obtained from the Directory.

The Directory Service Agent (DSA) : The DSA provides the database element of the Directory Service. The DSA receives the requests from the DUAs and if it has the access to the required information, sends the information back to the DUA concerned. If it does not have access to the information, it may pass the request to an other DSA, or may send back to the DUA where to find the required information.

3. Security in the Messaging System

The security architecture specified in this paper has been developed as part of the LOCATOR collaborative project, which is itself part of the Mobile Information Systems project, a major demonstrator within the UK Government-sponsored Alvey Programme. Here we only briefly outline the relevant features of the security architecture which are required in our formal specification work. A detailed description of the security architecture developed by the LOCATOR Project team is given in ([3]).

The architecture supports a number of security services, most of which are "end-to-end" in nature. These services are probably the most significant ones to the end users of a mail system. These end-to-end security services include : content confidentiality, message-origin authentication, content integrity, non-repudiation of origin, replay detection, and non-repudiation of delivery. There are also other services which are not end-to-end, such as access control on the User Agent/Message Store link. In our formal specification, we will only be concerned with the end-to-end security services. In Section 6.1, we briefly describe the type of access control service between the Message Store and the User Agent. Further we discuss the effect of the incorporation of Message Store on some of the end-to-end security services.

Before considering the security services in the messaging system in more detail, it will be useful to briefly describe some of the fundamental security mechanisms and concepts used in the provision of several of the security services.

3.1 Basic Security Mechanisms

Let us start by describing the structure of a message in this system. A message consists of two parts, namely an Envelope and a Content. The Envelope contains the necessary information for the message to be routed to the appropriate receivers and the Content is the actual information which is to be transferred.

3.1.1 Encryption and Key Management

Encryption is a fundamental mechanism which is required in the provision of several of the services. In our architecture, we use both symmetric and asymmetric (public key) cryptosystems. The symmetric encryption technique is used by the User Agent to encrypt and decrypt the message content. We have used the Data Encryption Standard (DES) for this purpose ([7]). The Cipher Block Chaining (CBC) mode ([8]) has been employed and the key and the initialization vector required for this mode are generated locally within the User Agent.

We will see in our formal specification that it is not necessary to go into detail regarding the implementation issues of the algorithm or the key generation process, as long as we can formulate the necessary properties of the algorithm and the paramaters to be generated. This issue of abstracting away from implementation details is a general one and it helps the designer to concentrate on the required generic properties of the system.

The management of the cryptographic keys of DES is done by employing asymmetric cryptosystems. In the LOCATOR system, the RSA public key cryptosystem ([9]) is used. Tne public key of the receiving User Agent is used to encrypt the DES key employed in message encryption. The sending User Agent transfers this encrypted DES key to the receiving User Agent.

For this technique to work, it is necessary to provide a guarantee to the sending user that the public key of the receiver is the "correct" one. The X.509 Authentication Framework ([10]) has been used in the authentication of public keys of the users. Each user's public key is stored in the Directory and a user wishing to have a secure exchange of messages with another user obtains the other user's public key using the Directory Service and then uses this key within the required security service. However as the Directory is not a trusted service, the user needs to verify the public keys obtained from the Directory. The X.509 Authentication Framework achieves this using off-line trusted entities called Certification Authorities (CAs).

3.1.2 Certification Authorities and Certificates

Let us now briefly consider the role of the Certification Authorities (CAs).

Each user must have a CA which he or she can trust and each CA has a public-key/secret-key pair. It is assumed that a user and his/her CA exchange their public keys in a secure and trusted manner. The role of a CA is to generate a Certificate for the users. It produces the certificate by signing a collection of information, including the user's name and the public key. The process of signing involves first hashing the information, using the hash function suggested in the Annex of X.509 ([10]), and then encrypting the hashed information using the RSA system under the control of the secret key of the signer. More specifically, the certificate of a user with the name A, produced by the Certification Authority CA, has the following form : <CA's name, user's name, user's public key, validity of users's public key>. The validity of the user's public key is specified using two dates, the first and the last dates on which the certificate is valid. This set of data together with the signature constitutes the Certificate, which is stored in the user's directory entry. The Certificate of A is denoted as $CA << A >>$.

Any user can check the public key of a given user A, by obtaining a trusted copy of the public key of A's Certification Authority and then using this key to check the signature on A's Certificate.

However, in a more general case, a user B wishing to communicate with user A may not know the public key of the Certification Authority of the user A. To deal with such cases, the notion of Certification Path has been introduced. The list of Certificates needed to allow a particular user to check the public key of another, is called a Certification Path. Each item in the list is a certificate of the next Certification Authority in the path between the users concerned.

3.1.3 Token

Another mechanism that is fundamental to the provision of security services in the messaging architecture is the "token". Any message which involves end-to-end security services requires the sender of the message to generate one or more tokens. A token, in general, consists of a number of parameters such as the encrypted data, the signed data, the name of the receiver, a timestamp and the identifiers of the signing and encrypting algorithms. Depending on the security service required some of these parameters may not be used by the receiver of the message.

The encrypted data and the signed data contain security-related information, dependent on the security service provided. For instance, the encrypted data contains the RSA encrypted version of the DES key used in the confidentiality service. The information making up the signed data is not encrypted. The timestamp identifies the date and the time the token was generated. One token is generated for each recepient of the message and the token contains the name of the intended receiver.

A signature is also provided with every token. This is generated by hashing all the data within the token using a modular squaring hash function and then encrypting the hashed value using the RSA secret key of the message sender. This is used to prove to the receiver that the contents of the token are not altered and that it comes from the claimed sender.

Whether all these parameters of the token are required is dependent on the type of security service required. For instance, if only the confidentiality service is required, it is not necessary for the receiver to check the token signature.

3.2 Security Services in the Messaging System

Let us now consider each of the security services in the messaging system in more detail.

3.2.1 Content-Confidentiality

The content-confidentiality service is provided by encrypting the contents of the message using the DES algorithm. The key and the initialisation vector required for the DES algorithm are generated locally by the sender of the message. In particular, a new key is generated for each message which needs to be protected.

Hence, if this security service is required, then the sending User Agent enciphers the content of the message and then encrypts the DES key used with the public key of the intended recipient. This encrypted key forms a part of the message token sent to the receiving User Agent via the Message Transfer System. To inform the receiver that message encryption has been used, an identifier is included which indicates the encryption algorithm used. In our case, this will indicate that the DES has been used. Furthermore, if the same message is to be sent to more than one user, the sending UA needs to produce more than one token using the appropriate public keys of the recipients.

The User Agent also needs to send the Certificate of the sending user which is obtained from the Certification Authority associated with that user. In our specification we assume that the certificate is actually stored in the User Agent. The sending UA also sends the Certification Path to the receiving UA. As explained earlier, this is necessary in order for the receiving UA to obtain the public key of the trusted Certification Authority which is then used to validate the public key of the sender of the message.

The system requires that this security service be provided to either all the receivers of the message, or to none of them.

The recipient of the message first checks to see if the message token argument is present in the message envelope. If so, this indicates that the end-to-end security services have been used and hence the recipient checks for the presence of the algorithm identifier to see whether message encryption has been used. If so, the recipient recovers the DES key and the initialization vector by decrypting the encrypted-data part with his secret RSA key. Now the recipient can recover the message content using the DES key and the initialisation vector.

3.2.2 Authentication

In this Section, we consider the provision of the following three services: Message-origin authentication, Content-integrity and Non-repudiation of origin. All these three services are provided using essentially the same mechanisms. In fact, in our architecture, these services are "grouped" together as a single service and the user cannot request one of these services on its own. It has been designed in this way because it is very unlikely that the users of a mail system would want to have one of them without the others.

The message-origin-authentication service is provided by the existence of the message token. Recall that the message token contains a signature which uniquely identifies the origin of the message. However, this does not guarantee that there has been no modification of the message. In order to achieve this, an integrity check called the "content-integrity-check" is included in the signed-data part of the token.

The content-integrity-check (CIC) is generated by the sender and is sent to all the recipients of the message for whom the service is intended to be provided. The CIC contains two fields, namely the content-integrity-algorithm-identifier and the content-integrity-check itself. The algorithm used to compute the content-integrity-check is the same as the modular squaring hash function that has been used in the calculation of signatures in the tokens and the certificates.

The receiver of the message first checks the envelope to see if a message token argument is present which indicates the use of end-to-end security services. If the content-integrity-check argument is present, then this indicates that these three authentication services are provided.

The receiver obtains the trusted copy of the public key of the Certification Authority of the sender. In the case where the receiver does not know the Certification Authority of the sender, he uses the certification path that has been supplied as part of the originator's certificate, to determine the trusted copy of the receiver Certification Authority's public key. Using this, the receiver validates the signature on the originator certificate.

The validation of the token is quite similar to the one carried out for the originator certificate, except that the key used to check the token is the originator's public key rather than the CA's public key. A valid token indicates that the content-integrity-check has not been modified.

To check the content integrity, the receiver recalculates the content integrity value and compares it with the one received.

If these checks are valid, then these confirm the authenticity and the integrity of the message content as well as non-repudiation of message origin, since only the user having the RSA secret key of the sender (i.e. the sender himself) could have generated the token.

In contrast to the content-confidentiality service, these three services can be provided to any subset of the recipients of a message.

3.2.3 Replay Detection

This service is provided to a receiver, by including a message sequence number within the signed-data part of the message token for that recipient. This sequence number is unique only with respect to the two users concerned, namely the sender and the receiver. That is, each user maintains a list of other users with whom he/she has exchanged messages in the past. The entries in the lists contain information regarding the transmitted and the received messages to/from other users. The inclusion of the sequence number detects replays.

However, in practice, if a user wishes to have conversations with thousands of other users, there may be problems of storage. An alternative technique may be to use the timestamps in the message tokens to prevent replay.

3.2.4 Non-Repudiation of Delivery

Non-repudiation of delivery service is somewhat different from the other security services above in that the service is actually provided by the receiver. The sender of the message requests the receiver for this service, by including a proof-of-delivery-request flag as part of the signed-data in the message token to the receiver. The proof-of-delivery is computed as the signature (using the receiver's secret RSA key) on the unencrypted message-content and various delivery related parameters. The receiver then returns the proof-of-delivery together with his/her certificate to the sender of the message via the Report Delivery Service.

The sender obtains a trusted copy of the public key of the receiver and then validates the certificate and the proof-of-delivery. Since the proof-of-delivery could have only been calculated using the secret RSA key of the recepient, this method provides the non-repudiation of delivery service.

Note that the system allows the proof of delivery to be requested from only some of the recipients of the message. This is possible because a distinct token is being generated for each message recipient.

4. The Formal Description Technique LOTOS

Formal, or mathematical, approaches to describing computer systems are gaining in popularity. This is not merely a fashion, but rather has arisen out of the increasing complexity of such systems. It is no longer adequate to take a natural language specification of a system, and start implementing in some programming language. Firstly, due to the scale of the specifications, it is difficult to decide on its consistency and non-ambiguity. Secondly, there is no method for reasoning about the specification to check whether the required properties have been captured correctly.

In the world of international communications standards it is obviously important that specifications are unambiguous, consistent, and represent the intended system. There has been a growing interest from this community in using formal approaches for describing complex communications standards.

LOTOS (*Language Of Temporal Ordering Specification*) has been developed within ISO as a Formal Description Technique (FDT) for the specification of OSI protocols and services. Work began in 1981, and LOTOS reached International Standard status (ISO 8807) in late 1988.

The development of the language was done under the Esprit/SEDOS programme, as a European collaboration project, with major contributions from the University of Twente in the Netherlands.

4.1 Basic Concepts

The basic underlying concept in LOTOS is that of an event. As a specification language, LOTOS is useful for describing systems in terms of the events, or interactions, of the system. Reactive communications systems can intuitively be described in terms of the allowed sequences of events of such systems.

To get a feel for describing systems in terms of events, consider the usual telephone system. Before a user can talk with another user, a number of events must occur. Firstly the user must lift the receiver, and then dial a number which is uniquely associated with a given subscriber. The telephone system is a reactive system, in that it can be used at any time by a subscriber. Communication can only occur between two subscribers if a well-defined series of events are performed beforchand.

Not only sequential order of events but also concurrent or parallel behaviour can be modelled in LOTOS. In describing certain systems, it is not necessary, and often not desirable, to describe an explicit ordering of independent activities. The explicit ordering is often an implementation detail, and could be implemented in many different ways without affecting the overall system behaviour.

LOTOS is a structured language, similar to structured programming languages. A collection of events in LOTOS can be combined to form a process. Also processes can be combined (with events) to form other processes. The whole specification is really just one process, which is constructed from a number of (sub) processes.

Communication between parallel processes is via synchronisation on events. This synchronisation is not the same as sending a message and waiting for an acknowledgement, but rather it is synchronisation by two processes actually sharing the same event. Thus an event is an atomic, non-interruptable action. Two processes can only communicate if they participate in the same event. In the above telephone example, we have a user and the system. The user can pick up the phone, and the system can recognise that the phone is picked-up. The user (as a process), and the system can only communicate if they both participate in the "picking-up-the-phone" event.

Events occur at interaction points (or gates). Each process has a defined set of interaction points. Thus two processes can communicate via an event if they have a common interaction point where the event can occur. In the above example, the interaction point is the telephone itself.

LOTOS is based on two language concepts, namely that of describing a system in terms of events, and that of describing the data types and values in terms of sets operations, and equations. The former part is called the process part, the latter the (abstract) data type part. These will be discussed in more detail in the following sections.

4.2 The Process Part

In LOTOS there are a number of operators for combining expressions. These operators allow for a powerful mechanism for describing communicating concurrent systems. We shall not define all the operators in full detail, as this can be found in ([6]), but shall describe some of the operators and their expressive power.

The most primitive combinator is the action prefix. Thus if we have a LOTOS expression B, we can prefix it with an event 'a', written 'a;B'. The resulting expression can then participate in event 'a', and then will behave as the expression 'B'.

The choice operator '[]' is a fundamental part of the language. The expression 'a;b;stop [] c;d;stop' models a system that can perform an event 'a' and then an event 'b' and then stop, or alternatively perform an event 'c' and then an event 'd' and then stop. This representation of alternative behaviours is often necessary as it is generally impossible to determine the next input in a reactive system (as this involves determining the behaviour of independent systems).

LOTOS has three different operators for combining expressions in parallel. The interleaving parallel operator is used to model independent processes or expressions. This is useful for modelling independent aspects of a syst m, such as independent functions of a given entity, or independent entities in a networked system. The notation ||| is used to represent this operator; for instance, B_1 ||| B_2 implies that expression B_1 is independent of expression B_2.

The second parallel operator is the synchronous parallel operator, with notation B_1 || B_2. B_1 and B_2 are fully dependent on each other: any action that B_1 participates in must also be shared by B_2, and vice versa. If B_1 can only participate in an event that B_2 cannot participate in (or vice versa), then the whole expression cannot participate in any event, and the expression represents deadlock. This operator is useful for defining composite restrictions on the possible events of a system.

The third parallel operator is a combination of the other two. Its intention is that two processes must synchronise on som events, but must be independent on all others.

Other operators include enabling (for sequential composition of processes), disabling (for the interruption of processes), and hiding (for masking some events of the system from the environment of the system). Hiding is useful for defining internal behaviour which cannot be observed through the system interface.

4.3 The Data Type Part

So far we have only considered the notion of event. We have not yet described *what* is communicated on an event - i.e. the data values.

Orthogonal to the language of events in LOTOS, is the language for describing data types and values. The particular abstract data type language (or equational algebra) used in LOTOS is ACT ONE. A *type* in the language can be considered as a module for defining a number of sets, or *sorts*, and operations on these sorts.

To define a sort, one gives the sort name, and any number of operations on the sorts. The operations (or functions) are defined by giving the operation name, a list of names of the domain sorts, and the range sort. Furthermore, equations can be given which define constraints on the operations (hence the name equational algebras).

For an example, let us consider the ACT ONE data type Boolean. (LOTOS keywords are in bold type.)

```
type     Boolean is
sorts    Bool
opns     true      : ->       Bool
         false     : ->       Bool
         not       : Bool ->  Bool
eqns     ofsort    Bool
         not(true) = false;
         not(false) = true;
endtype
```

Here we have a whole data type module called 'Boolean'. It consists of one sort, with the name 'Bool'. Three operations are defined, 'true', 'false', and 'not'. The operations 'true' and 'false' are essentially constants of the sort 'Bool'.

The operation 'not' is defined to be an operation, or function, mapping elements of sort 'Bool', onto elements of sort 'Bool'. The equations are defined to be equations with values of sort 'Bool'. It is necessary to define the sort of the equations, as overloading of operators is allowed in the language. The equations define the value of the function 'not' applied to the constants 'true' and 'false'. Without these equations 'not(true)' would not have the same value as 'false'.

4.4 Events, Values and Gates

In Section 4.1 we briefly mentioned gates, or interaction points. When defining a process, we must define the interaction points of that process. The events of a process are defined in terms of these interaction points. Thus, a simple event which does not represent the transmission of values, but just the communication between processes, is written by giving the interaction point of that event. Hence the name of a simple event is synonymous with its interaction point.

However, to describe the communication of values between processes by events, the events must be described in a more complicated fashion. An event, therefore, is described by giving the interaction point, and a list of event offers.

An event offer consists of either a particular value (such as 'not(true)'), or a parameterised value of a given sort (such as 'x:Bool'). Event offers are prefixed with either '!' or '?' to represent the offering or accepting of values, respectively. For example, 'g ! not(false)' describes an event at interaction point 'g', which offers a value 'not(false)'.

Going back to the synchronization operator, the expression

'g ! true ; stop || g ? x:Bool; stop'

represents a system that can offer the value 'true' at gate 'g' and then stop, which must synchronise with a system that will accept any value of sort 'Bool' at gate 'g' and then stop. Synchronisation can occur, as they both offer an event at the same interaction point, and the event offers have a valid correspondence. After the event, the parameter 'x' is assigned the value 'true'.

However, the expression

'g ! true ; stop || g ! false ; stop'

behaves as deadlock, as the two subexpressions cannot communicate. Although they offer values at the same interaction point, they are not the same value, and therefore cannot be seen as a single event. Similarly for

'g ! true ; stop || g ? x:Nat ; stop'

The expression must deadlock as the value 'true' is not of the sort 'Nat', and therefore the events offered by the subexpressions are not the same.

5. LOTOS Specification of Security Services

In this paper, we only describe the specification of those services which involve the provisi n of security in the messaging system.

The LOTOS specification consists of two parts, namely the service specification and the protocol specification. Note that the use of the word *"service"* here is somewhat different from the one used in *"security services"* above. Let us first briefly describe the essential difference between the service and protocol specification.

A system provides a set of *"services"* which allow various interactions between the users in the system. The *"protocols"* are mechanisms which provide the services. Hence the user is concerned with the nature of the services but not with how the protocol manages to provide them.

A service specification is really a specification of the requirements of the system. It can be considered as a very high-level, or *"abstract"* description of the system. There should be no detail as to how the system may intend to implement these services. Thus it can be seen as an interface description of the system, with the only detail being that which the user *sees*.

On the other hand, a protocol specification is a description of the mechanisms which should be used by the system to provide the service. However, a protocol specification can still be considered abstract in some sense. For international standards, it is important that protocol specifications should not give unnecessary detail, as this may favour some companies' hardware/processor offerings over others. A protocol specification may say, for example, "receive value x from user, make a copy, increment counter, and pass x to another user". This does not say how to copy the value, or increment the counter.

Security aspects are modelled as part of both service and protocol specifications. That is, each specification is a combination of process definitions, and also data type definitions. Note that the term "model" is used in a generic sense and it should not be confused with the security models as given in the Trusted Computer System Evaluation Criteria ([11]). The complete specification consists of some thirty process and some forty data type definitions ([12]).

5.1 Service Specification

As mentioned previously, a service description must not contain internal details of the system, but must only specify the behaviour of the system in terms of the behaviour at the system interface. All events in the service specification must therefore take place at the interaction points visible to the user.

In our description we have only used one interaction point, which is called "user". There are a number of constraints needed on all events of the system, called global constraints. It is for this reason that only one event format is used (so that the global constraints have the correct event format for every event of the system). The event format is

'user ? user-id:Name ? user-op:UserInteraction'

Thus every event occurs at gate "user", has a user-identifier of sort "Name", and a user operation of sort "UserInteraction". The user-identifier identifies which user of the system the particular event is related to, and the user operation is the service primitive and associated parameters of the particular event.

Strictly speaking, our system can only have one user interaction at any instant. However, this is very reasonable as a model, since all events, or interactions are atomic. So, if any two independent users want to use the system at the same time, it is reasonable to assume that one of them uses the system just before the other (but without necessarily saying which). Such an interleaving model of concurrency forms an important element of LOTOS semantics.

5.1.1 Modelling Security Services

From the viewpoint of the service, the security services are primarily of the following form : a request is made for a specific transformation of message, and such an appropriate transformation of the message is then delivered. There is no need to define details of the algorithms and the functions used in the security services. However, we must define the properties of the security services.

Firstly, the security services are requested as part of the service primitives of the system. Thus it is necessary to model the security services as data types and values. When a certain service primitive is requested, the associated security services can also be requested in the form of parameters of the service primitive.

The security services are provided by some forms of manipulation of given data, and also by the addition of extra security parameters. For example, a request may be to send some data D, encrypted using some given key K, to some user U. The value received by U could be represented by *"encrypt(D,K)"*, representing encrypted data D under key K. Note that no mechanism is given for performing the encryption process.

User U cannot access the data D without decrypting the encrypted message. To be able to do this, an appropriate decrypting function must be defined, and also some rules (equations) must be given for defining how the decryption and encryption functions are related. So the equation

decrypt(encrypt(D,K),K) = D

constrains the decryption function to return the initial value of an encrypted data message D, provided the same key K has been used. This is a generic property of a symmetric encryption function.

5.1.2 Examples from the Service Specification

In this section we give two examples of LOTOS data types defined in the service specification, to model security aspects of the service. These examples should be sufficient to gain an understanding of the modelling power of the language, and also of the application of the language to specific problems.

```
type    SM-options      is Boolean, SetOfUser, Message
sorts   SM-option
opns    MakeSMOption :      Bool, SetOfUser, SetOfUser
                            Message, SetOfUser
                                      -> SM-option
        ContConf        :SM-option    ->Bool
        PoD             :SM-option    ->SetOfUser
        DataAuth        :SM-option    ->SetOfUser
        Message         :SM-option    ->Message
        Recipients      :SM-option    ->SetOfUser
        _eq_,_ne_       :SM-option,SM-option
                                      ->Bool

eqns    forall cc,c1:Bool, pd,da,rc,p1,d1,r1:SetOfUser,
                mg,m1:Message, sm1,sm2:SM-option
        ofsort Bool
        MakeSMOption(cc,pd,da,mg,rc) eq MakeSMOption(c1,p1,d1,m1,r1) =
             (cc eq c1) and (pd eq p1) and (da eq d1) and (mg eq m1) and (rc eq r1);
        sm1 ne sm2 = not(sm1 eq sm2);

        ContConf(MakeSMOption(cc,pd,da,mg,rc)) = cc;

        ofsort SetOfUser
        PoD(MakeSMOption(cc,pd,da,mg,rc)) = pd;
        DataAuth(MakeSMOption(cc,pd,da,mg,rc)) = da;
        Recipients(MakeSMOption(cc,pd,da,mg,rc)) = rc,
        ofsort Message
        Message(MakeSMOption(cc,pd,da,mg,rc)) = mg.
endtype
```

Here we define the type 'SM-options', which includes, or inherits, the types 'Boolean', 'SetOfUser', and 'Message'. A new sort 'SM-option' is introduced. A number of operations are defined from elements of the sort 'SM-option'. The operation 'MakeSMOption' maps elements from other sorts onto the set 'SM-option'. The set "SM-option" is constructed from elements of other sets, with the constructor being the 'MakeSMOption' operation.

The operations '_eq_' and '_neq_' are infix operations defining equality on elements of the new set 'SM-option'.

The operations 'ContConf', 'PoD', 'DataAuth', 'Recipients', and 'Message' are defined (by the equations) to extract individual parameter values of the composite elements of the set 'SM-option'.

Secondly, we give the type 'Keys' which defines the encryption and decryption keys used by the encryption and decryption functions and the hashing function.

```
type     Keys is           NaturalNumber, Boolean, Message, Name
sorts    secret-key, public-key, symmetric-key
opns     symmetric-key  : Nat, Nat             -> symmetric-key
         secret-key     : Nat, Nat, Nat        -> secret-key
         public-key     : Nat, Nat             -> public-key
         _corresponds-to_: public-key, secret-key   -> Bool
         Modulus        : secret-key           -> Nat
         Modulus        : public-key           -> Nat
 eqns    forall    n1,n2,se,pe,mod:Nat

ofsort   Bool
         n ne 0 =>
         public-key(pe,mod) corresponds-to secret-key(se,n1,n2) =
              (mod eq n1*n2) and (se*pe eq succ(0)+
                        n*(n1-succ(0))*(n2-succ(0)));

ofsort   Nat
         Modulus(secret-key(se,n1,n2))= n1*n2;
         Modulus(public-key(pe,mod))= mod;
endtype
```

We have removed the equality relations on elements of the new sorts, to make things simpler to read. The remaining operations are essential to the modelling of security in the service.

The operations 'symmetric-key', 'secret-key', and 'public-key' map lists of natural numbers onto the newly defined sets. These are constructor operations for these sets. The operation 'corresponds-to' is a comparison operation or relation between public and secret keys. The two 'Modulus' operations "extract" the modulus number associated to the elements of sort 'secret-key' and 'public-key'. As mentioned in Section 2, the DES symmetric key cryptosystem and the RSA public key cryptosystem have been used.

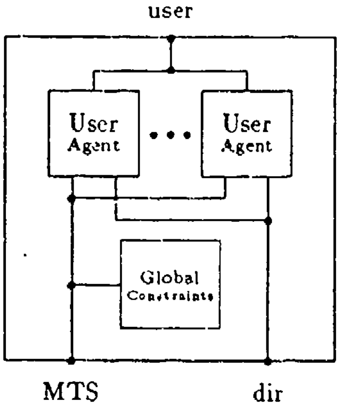


Figure 2: Structure of the specification

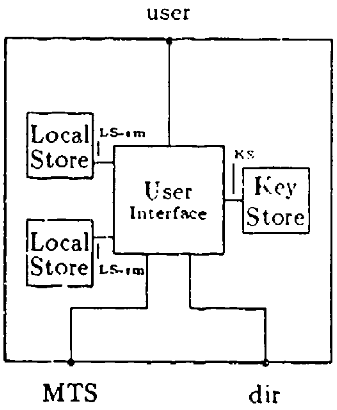


Figure 3: Structure of the process "UserAgent"

5.2 Protocol Specification

The protocol specification describes the mechanisms used by an entity to provide the services described in the service specification (Section 5.1). We first give a brief description of the overall structure of the protocol specification and then consider some examples to illustrate how some of the security aspects are modelled within the protocol.

5.2.1 Structure

As in the case of the service specification, the interactions between the User Agents are modelled via a single gate "user". In addition, to describe the protocol, we need to model the interactions between the User Agents and the Message Transfer System and the Directory. The interactions between a User Agent and the MTS occur via the gate "MTS" and the interactions between the User Agent and the Directory occur via the gate "dir". A diagram representing the structure of the protocol specification is given in Figure 2.

Each user has a corresponding User Agent process and each User Agent process is described as an interleaved composition of the following four processes (see Figure 3) :

- UserInterface Process which specifies the complete protocol;
- KeyStore Process which is used to store the user's secret key of the public key cryptosystem;
- LocalStore1 Process is connected to the UserInterface process through the gate "LS-rm" and it stores the messages received by the UserInterface;
- LocalStore2 Process is connected to the process UserInterface through the gate "LS-sm" and it stores the messages sent by the UserInterface process when the proof of delivery is

required.

Two instantiations of the process LocalStore help to avoid problems in synchronising the "sending part" and the "receiving part" before every update.

The process UserInterface is a sequential composition of two processes : ConnectToMTS and Operation. As the name implies the process ConnectToMTS connects a user with the MTS and checks whether the connection is well-established. The process Operation performs the services requested by a user and also receives messages and proof-of-delivery from the MTS.

System failure or transmission error is modelled by the process DisconnectFromMTS which may happen at any time and in this case, a new attempt is made to connect to the MTS.

The process Operation has three interleaved processes :

- the StoreOperation process stores the secret key in the KeyStore (cf service Store Secret Key);
- the process OutputOperation controls the message sending part between the User Agent and the MTS; and
- the process InputOperation controls the receiving part between the User Agent and the MTS.

We can now proceed to consider some examples of process specifications to illustrate the modelling of security aspects within the protocol specification.

5.2.2 Modelling Security in Protocol Specification

In general, when modelling security in a protocol specification, the designer has some amount of freedom in deciding as to how much of the security aspects are incorporated in the process specifications and how much are defined as part of the data types. This occurs for instance when an operation can be either specified as part of the opns of the data type definition or can be included within a process as a computation.

As an example, let us consider a process which receives some data, and then outputs a hashed and encrypted copy of this data. This could be modelled by the following process.

```
process  Hash-Encrypt [gate]: exit :=
         gate ? x:Data ;
         gate ! hash(encrypt(x));
         exit
endproc
```

The process 'Hash-Encrypt' gets an element of sort 'Data', assignes it to the parameter 'x', and then outputs a hashed and encrypted copy of 'x'. This hashing and encrpyting is done in one single action. An alternative representation could be the following:

```
hide mid in
         Encrypt[gate,mid] |[mid]| Hash[mid,gate]
where
```

```
process  Encrypt[gate,mid]: exit :=
         gate ? x:Data;
         mid ! encrypt(x);
         exit
endproc

process  Hash[mid,gate]: exit :=
         mid ? x:Data;
         gate ! hash(x);
         exit
endproc
```

In the above, two separate processes are identified as performing the encryption and hashing functions. The process 'Encrypt' gets some data, and then outputs to the process 'Hash' an encrypted version of the data. 'Hash' receives some data (which happens to be encrypted), and outputs a hashed version of what it received.

The event on interaction point 'mid' is hidden from the user of these processes. As far as the user is concerned these two representations look exactly the same. The second version has two processes performing separate tasks of hashing and encryption whereas the former version identifies only one process with both the hashing and encryption operations.

Let us now consider some examples of LOTOS data types and processes used in the protocol specification in modelling security properties.

First we outline the definitions of two data types: Certificate and Token. Recall that we described these two concepts in Sections 3.1.2 and 3.1.3 respectively. So it is useful to look at their data type definitions and compare them with the descriptions given in Sections 3.1.2 and 3.1.3. Not all the equations are included, but the illustration should still be valid.

```
type   Certificate is Time, Name, AlgorithmType, Keys, Boolean,
       Message, Encrypt-Hash
sorts  Certificate, NonSignedCert

opns   construct-non-s-cert: AlgType, Name, Time, Time, public-key
                                                  ->NonSignedCert
       compute-cert: secret-key, Name, Time, Time, public-key
                                                  ->Certificate
       construct-cert: NonSignedCert, Message     ->Certificate
       signature         : Certificate            -> Message
       non-s-cert        : Certificate            -> NonSignedCert
       subject-name      : Certificate            -> Name
       start-validity    : Certificate            -> Time
       expiry-time       : Certificate            -> Time
       public-key        : Certificate            -> public-key
       certificate-valid : Certificate, Time      -> Bool
       check-certificate: Certificate, Time, public-key -> Bool
       convert-nscert-message   : NonSignedCert   -> Message
       convert-message-nscert   : Message         -> NonSignedCert

eqns   forall nsc: NonSignedCert, mg: Message ...
```

```
          ofsort NonSignedCert
          convert-message-nscert(convert-nscert-message(nsc)) = nsc;
          non-s-cert(construct-cert(nsc,mg)) = nsc;
          ⋮
endtype   (* Certificate *)
```

First recall that the construction of a certificate for a user is performed by the Directory. We construct the Certificate using two parts, namely a non-signed contents part and a signature part. The operation "construct-cert" performs this function. The Directory uses its secret key to sign the hashed form of the contents part. This is done as part of the "compute-cert" operation. The contents part is built from its various components as described in Section 3.1.2 using the operation "construct-non-s-cert" operation.

The other operations given in the definition of the type are self-explanatory. It can be easily seen from the example that the equations describe the required properties of the functions specified in the **opns** part.

A similar approach has been used in the specification of the data type *Token* given below. That is, a token is assumed to be composed of two parts: a non-signed part and a signature. The operation "construct-token" constructs a token from these two parts. The signing of the contents part of the token is done as part of the "compute-token" operation. The non-signed contents part is built from its various components as described in Section 3.1.3 using the "construct-non-s-token" operation.

The other operations and the equations given in the specification are again self-explanatory.

```
type      Token is Time, Name, AlgorithmType, Keys,
                    Boolean, Message, Encrypt-Hash
sorts     Token, NonSignedToken
opns      construct-non-s-token: AlgType, Name, Time, AlgType, Message,
                          Bool, AlgType, Message  -> NonSignedToken
          compute-token: secret-key, Name, Time, AlgType, Message,
                          Bool, AlgType, Message  -> Token
          construct-token : NonSignedToken, Message      -> Token
          signature       : Token                       -> Message
          non-s-token     : Token                       -> NonSignedToken
          recipient-name  : Token                       -> Name
          CIC-alg-type    : Token                       -> AlgType
          cont-int-check  : Token                       -> Message
          proof-of-delivery: Token                      -> Bool
          enc-alg-type    : Token                       -> AlgType
          encrypted-token: Token                        -> Message
          check-token     : Token, public-key           -> Bool
          convert-message-nstoken   : Message               -> NonSignedToken
          convert-nstoken-message   : NonSignedToken        -> Message

eqns      forall nst: NonSignedToken, mg: Message ...
          ofsort NonSignedToken
          convert-message-nstoken(convert-nstoken-message(nst)) = nst;
          non-s-token(construct-token(nst,mg)) = nst;
```

```
        :
endtype (* Token *)
```

Let us now consider some examples of modelling security aspects within process specifications. We will describe two processes, one from the set of Input Operations and the other from the set of Output Operations.

Example 1 : Process - Receive Message from the MTS

The process Receive Message from the MTS is one of the subprocesses associated with the process InputOperation. More precisely, the process InputOperation is specified as a choice between the three user services - List Message, Read Message and Delete Message - and the MTS service Receive Message.

The overall function of this process is to receive an "envelope" from the MTS and to check its various components and perform the appropriate actions. Let us go through the specification of the Receive-Message process step by step and see how the security-relevant operations defined in the abstract data types are being used.

The process first checks whether the envelope is valid. An envelope is defined to be valid if the certificates and the token in the envelope are valid. This is done using the function

"check-envelope((env),ctime,dir-key)"

Note that the definition of "check-envelope" is not given in the process specification but is defined as part of a data type definition elsewhere as being equal to

"check-certificate(origin-cert(env), t, p-key) and
check-certificate(recep-cert(env), t, p-key) and
check-token(token(env), p-key)".

The process then checks whether a proof-of-delivery has been requested, using the function

"proof-of-delivery(token(env))".

If the envelope is valid and if the confidentiality service has been used, then the process recovers the symmetric key from the token and decrypts the message and stores it in the received message local store LS-rm. If the proof-of-delivery has also been requested then it is computed using the function "compute-ProofOfDelivery(s-key,env))" and sent to the MTS.

The complete specification of the process Receive-Message is given below:

```
process Receive-Message[user,MTS,LS-rm,KS]
          (user-id:Name, dir-key:public-key, s-key:secret-key) : noexit :=
     choice env:Envelope []
          MTS ! user-id ! message-delivery(env);
          choice ctime:Time []
                    [iscurrenttime(ctime)] -> i;
               let envelope-correct:Bool = check-envelope(env,ctime,dir-key),
                    proof-required:Bool = proof-of-delivery(message-token(env)) in
          (
               [not(envelope-correct)] ->
```

```
            MTS ! user-id ! message-delivery-result(empty-res);
            InputOperation[user,MTS,LS-rm,KS] (user-id,dir key)
         []
         [envelope-correct] ->
            let sd:Name = subject-name(originator-cert(env)) in
               LS-rm ! user-id !LS-update
                  ! MessageEl(sd,message-decrypt(s-key,env),false);
         (
            [not(proof-required)] ->
               MTS ! user-id
                  ! message-delivery-result(empty-result);
               InputOperation[user,MTS,LS-rm,KS] (user-id,dir-key)
            []
            [proof-required] ->
               MTS ! user-id
                  !message-delivery-result(compute-ProofOfDelivery(s-key,env),
               InputOperation[user,MTS,LS-rm,KS] (user-id,dir-key)
         )
      )
endproc (*Receive-Message*)
```

Example 2 : Process Submit Message.

The process Submit Message is one of the subprocesses associated with the process OutputOperation. More precisely, the process OutputOperation provides a choice between the two user services - "Submit Message" and "Confirm Proof Of Delivery"- and the MTS service Receive Proof and a Timeout process.

The process Submit Message describes the protocol associated with the "Submit-Message" service described in Section 5.1. This process checks whether the content-confidentiality service is required. If it is the case, then a symmetric key (the DES key together with the initialisation vector) is generated. The following LOTOS construct specifies the selection of a symmetric key

"choice key : symmetric-key [] i".

For each message recipient requiring security services, a trusted copy of the certificate is obtained from the Directory. Envelope is computed using the following parameters : the secret RSA key of the user constructing the envelope, encrypted or plain message, the arguments of the token and the originator's and recipient's certificate. Note that within the process specification, this is achieved using the "compute-envelope" function. If the proof-of-delivery has been requested then the message submitted is stored in the local store LS-sm, which will be used in validating the receipt of the proof-of-delivery.

```
process SM-operation [user,MTS,dir,LS-rm,KS]
      (user-id:Name, dir-key:public-key, s-key:secret-key) : noexit :=
   choice op:SM-option []
      user ! user-id ! SM-request(op);
      dir ! user-id ? orig-cert:Certificate;
         let mod : Nat = Modulus(public-key(orig-cert)),
            mg : Message = Message(op), rc : SetOfUser = Recipents(op) in
      (
                [not(ContConf(op))] ->
                        SM-op-1 [user,MTS,dir,LS-sm,KS]
                                (user-id,dir-key,rc,hash(mod,mg),op,no-alg-type,
                                        no-alg-type,empty-message,mg,orig-cert,s-key)
                []
                [ContConf(op)] ->
                (
                        choice key : symmetric-key [] i;
                                SM-op-1 [user,MTS,dir,LS-sm,KS]
                                        (user-id,dir-key,rc,hash(mod,mg),op,asymmetric-alg-type
                                        asymmetric-alg-type,asymmetric-encrypt(public-key(rec-cert),
                                        convert-symmkey-message(key)), symmetric-encrypt(key,mg),
                                        orig-cert,s-key)
                )
        )
endproc (*SM-operation*)
```

6. Discussion

6.1 Further Security Issues in the Messaging System

In our specification we have not included the Message Store component of the Messaging System. The functions of the Message Store are quite important, particularly in the case of a Mobile System. The next step is to include the interactions between the Message Store, Message Transfer Agent and the User Agent. From a security point of view this results in several additional interesting issues some of which we now describe.

Recall that in Section 3, we mentioned that there are also other services which are not end-to-end in nature ([3]). One such service is the "access control" service between the User Agent and the Message Store. Essentially this is achieved by using another type of token called a "bind-token" which is exchanged between the User Agent and the Message Store at the time of connection initiation. Again the token includes such information as signed-data and time which are then checked by the Message Store to determine if the request is valid. The MS then returns a token to the UA which makes further checks and if all these checks are satisfied, then the connection can be established. We can easily include this service in our specification without any difficulty.

However, the inclusion of the Message Store does pose a problem with non-repudiation of delivery service. This is because the end point of the message delivery system now becomes the Message Store rather than the User Agent. This in turn implies that the required proof-of-delivery needs to be computed by the Message Store and hence the Message Store must know or have access to the RSA keys. Different methods have been devised ([3]) to provide partial solutions to this problem. We will not describe these methods here and the interested readers should refer to ([3]). It is sufficient to say that although there may not be an ideal solution, one can still provide secure messaging.

It is also worth mentioning that there may be problems with the form of tokens as defined in the X.400 and X.500 Recommendations. Without going into detail, the problem arises due to the fact that the token involves signing of encrypted information. This method only guarantees the authenticity of the *encrypted* information rather than the corresponding plain data. There are situations where this does pose a problem and again some modifications can be made which will overcome this deficiency. These issues have been explored in ([3]). It is worth pointing out that such problems do not arise in the LOCATOR architecture because of the way the *content-integrity-check* (CIC) has been calculated and used ([3]).

6.2 Suitability of LOTOS for modelling security

LOTOS was designed as a specification language, not an implementation language. Thus the specification is at a very high level, and is more concerned with the properties of the security functions, rather than of the algorithms used to implement such functions.

In discussing the LOCATOR Project, we mentioned the use of RSA and DES algorithms. As it can be seen from our specifications, the particular algorithms are not defined. However, the really important aspects of these algorithms are defined in terms of equations constraining operations.

As mentioned previously, the underlying description mechanism of LOTOS is that of events. It is interesting to note that much of the security modelling is done using the data type part of the language.

In the service specificiation we saw that all of the security services were defined using abstract data types. This does not mean that LOTOS is not suited to modelling security services. but rather that security services are very strongly connected with data values.

It is very difficult to reason "rigorously" about general properties of the system. For this kind of formal reasoning, a logic based language is best suited. However, our approach is useful for animating the design, well before implementation takes place. This is particularly useful from the designer's point of view as system errors can be detected and rectified at the design stage prior to implementation. Thus with a LOTOS specification it is possible to symbolically execute (simulate) the specification. Many of the design errors could be detected at this stage.

6.2.1 Tools

The development of tools supporting the specification and analysis process is an essential part of the propagation of formal methods. To gain wide acceptance, it is necessary to present formal languages in a way which is amenable to the software eng[illegible] Much of the complex mathematical detail needs to be hidden, a user-friendly approach to t[illegible]age must be taken, and as much automated support needs to be given.

The language LOTOS is a structured language, using the conventional (ASCII) character set. However, the operators are not obvious, and large specifications are usually difficult to read. There is currently development in ISO to create a graphical representation for LOTOS, G-LOTOS ([13]), which should make it easier to read and appreciate the structural relations.

LOTOS is being used more and more and consequently, there is a growing knowledge of how to use the language, with a number of styles appearing. However, there is as yet no methodolcgy.

As people are becoming interested in LOTOS, so there is a growing interest in the development of supporting tools. One of the first tools, called HIPPO, was developed under a European ESPRIT Project, sponsored by the EC, and is commercially available.

At Hewlett-Packard, we have developed our own tool SPIDER ([14]). This is an extensible tool, which currently consists of a syntax checker, static semantics checker, and a graphical simulator. Work is also progressing on the development of a demonstrator tool for editing G-LOTOS specifications. There are further plans to extend this tool by adding automatic generation of test suites, and compilation of LOTOS specifications.

7. Conclusions

In this paper, we have described a formal specification of a security architecture for a distributed message handling system.

The messaging system considered is the CCITT's Message Handling System. The security architecture has been developed as part of the LOCATOR collaborative project within the UK Government Alvey Programme. The security services provided include content confidentiality, integrity, authentication and non-repudiation. These security services and the associated mechanisms in the architecture have been formally specified using the Formal Description Technique LOTOS.

The formal specification has proven to be useful in many respects. It has allowed us to isolate and model only the security issues involved in the design of the messaging system. We have illustrated such modelling of security aspects using some small examples taken from the full specification. The specification has provided the necessary abstraction allowing representation of architectural aspects and the hiding of implementation details. This investigation has further enabled us to assess the use of LOTOS in the specification of a practical yet reasonably large system. We hope to continue this work in the future by carrying out the simulation of the specification using the LOTOS toolset SPIDER which is currently being developed.

8. References

[1] C.C.I.T.T. *Draft Recommendations X.400 : Message handling Systems - System and Service Overview*, Version 5.5, April 1988.

[2] I.S.O. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model. Part 2 : Security Architecture 7498 -2*, International Standards Organization 1988.

[3] C.J.Mitchell, P.D.C.Rush and M.Walker, *A secure messaging architecture implementing the X.400-1988 security features*, Tech. Memo No. HPL-ISC-TM-88-076, HP Labs., Bristol, UK, Nov.1988.

[4] C.C.I.T.T. *The Directory X.500*, Final Draft Recommendations, March 1988.

[5] I.S.O. *Information Processing Systems -Open Systems Interconnection - LOTOS - A Formal Description technique based on the temporal Ordering of Observational Behaviour 8807*, International Standards Organization 1988.

[6] T.Bolognesi and E.Brinksma, *Introduction to the ISO Specification Language LOTOS* (Invited Paper), Proc. of First International Conference on Formal Description Techniques, Sept.1988.

[7] *Data Encryption Standard (DES)*, FIPS Publication 46, National Bureau of Standards, U.S. Dept. of Commerce, Washington DC, 1977.

[8] *DES Modes of Operation*, FIPS Publication 81, National Bureau of Standards, U.S. Dept. of Commerce, Washington DC, 1980

[9] R.L.Rivest, A.Shamir and L.Adleman, *A Method for obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol.21, 1978, pp120-126.

[10] C.C.I.T.T. *Draft Recommendations X.509 : The Directory - Authentication Framework*, Version 7, November 1987.

[11] US DoD, *DoD Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, Dec.1985.

[12] C.Calvelli, *LOTOS Specification of LOCATOR Security Aspects* HP Internal Report, 1988.

[13] I.S.O. *G-LOTOS : A Graphical syntax for LOTOS*, ISO/IEC JTC1/SC21 N3253, January 1989.

[14] S.Johnston, *SPIDER - Service and Protocol Interactive Development Environment*, Proc. of First International Conference on Formal Description Techniques, Sept.1988.

9. Acknowledgements

The authors would like to thank Claudio Calvelli, student from University of Pisa, Italy and members of the LOCATOR Project. The authors are also grateful to the anonymous Referees for their comments.

The SDOS System: A Secure Distributed Operating System Prototype *

Raymond Wong, Mathew Chacko, Eugene Ding,
Brian Kahn, Norman Proctor, John Sebes, Ram Varadarajan
Odyssey Research Associates
525 Middlefield Road, Suite 250
Menlo Park, CA 94025

Abstract

We describe the design requirements and the system architecture for the SDOS system which is an experimental prototype for a secure distributed operating system designed to meet TCSEC B3 requirements. Key design requirements include the ability to connect machines of heterogeneous hardware and software architectures, and the preservation of existing investments in machines and software applications. The object model is used as the basic structuring paradigm for the system design. Object managers implement abstract operations on object instances of a type. Clients access objects by invoking operations on objects. A simple message-passing Switch provides efficient communications between clients and managers. The design uses a layered implementation architecture with the SDOS Switch and object managers built on top of an off-the-shelf secure constituent operating system.

Introduction

The primary goal of the Secure Distributed Operating System (SDOS) development project is to advance the state of the art in the area of secure distributed operating systems. The SDOS system will be a prototype of a secure distributed operating system designed to meet DoD TCSEC B3 security and assurance requirements, [TCSEC 85]. The SDOS design borrows many of its abstractions and concepts from the Cronus distributed operating system developed by Bolt Beranek and Newman Inc., [Cronus 86]. For example, the basic object-oriented client/server model has been retained. However, the system architecture has been redesigned in order to provide TCSEC design assurance, multi-level security, enhanced identification and authentication, enhanced discretionary access control, configuration security, audit, and network protection.

In this paper we describe the design requirements and system architecture for the SDOS system. The paper is organized as follows. The next section discusses the design requirements. This includes a discussion of both functional and security requirements. The System Overview section presents the basic SDOS architecture that satisfies the previously identified requirements. The layered implementation architecture is presented. The Operation Invocation Scenarios section illustrates how the system processes typical operation invocations. Finally, the last section discusses the current status of the project and describes future plans.

*Funded by the U.S. Air Force, RADC contract F3062-88-C-0146

System Design Requirements

An operating system provides a set of powerful abstractions by which users may use, share, and control the resources of the underlying machine. A distributed operating system presents the user with a set of uniform abstractions for the resources at multiple, independent processing locations. The distributed operating system provides location transparency and makes the distributed system appear as a "virtual uniprocessor". A secure distributed operating system permits users to access objects only if they are consistent with a set of well-defined security policies. In addition, the secure distributed operating system may provide enhanced auditing and network protection. In this section we will outline the functional and security design requirements of the SDOS system.

Coherence and Uniformity

The SDOS system must provide a coherent and uniform integration of the distributed processing resources. System services must be available to the user through a uniform set of abstractions. Objects such as files, directories, processes, services and I/O devices must be accessed using a global naming facility and a uniform set of communication primitives.

Heterogeneity and Evolution

Many distributed systems have evolved through the interconnection of existing stand-alone machines of possibly different hardware and software architectures. These machines may be connected by a local-area network (LAN) at a specific location or by a wide-area network connecting LANs at different locations. The SDOS system should permit the interconnection of machines of differing architectures over different communication media in order to facilitate the sharing of information and computing resources between organizations, and to provide increased reliability and availability of services.

Reliability and Availability

The SDOS system should be reliable in the sense that the integrity of its data should be maintained even across system failures. The SDOS system should be available or be fault-tolerant so that services continue to be accessible even if parts of the system should fail.

Scalability

The SDOS system may be configured with different processing elements to accommodate a range of users and specific applications. It should be possible to incrementally expand the system with additional resources over time.

Preservation of Existing Applications

The SDOS system should permit the execution of existing applications such as compilers, editors, window systems, databases, etc. The design of the SDOS system should not require the re-coding of these common applications. In addition, it should be possible to permit SDOS users access to specialized computing resources that may be attached to the system such as high-speed parallel processors, special purpose symbolic processors, or high-speed graphics devices.

TCSEC Requirements

The SDOS system will be designed to meet the TCSEC B3 functionality and assurance requirements. Therefore, the sharing of information and resources on SDOS will be consistent with the enforcement of a mandatory security policy; enforcement of a discretionary access control policy; reliable identification and authentication of users and their processes; and auditing of user and system activity. There will be a single SDOS security administrator. A trusted path will exist for security-critical operations.

B3 assurance requirements require that SDOS have a formal security model, Detailed Top Level Specification (DTLS), covert channel analysis, various correspondences with the DTLS, and require that the system design minimize the Trusted Computing Base (TCB) and employ modularization and least privilege.

Trusted Network Interpretation Requirements

The network interconnecting the components of the SDOS system must provide message integrity, protection from compromise, and protection from denial of service, [TNI 87].

System Configuration

The present SDOS design is aimed at providing support for the connection of multiple SDOS hosts on a single local area network. In addition, single level untrusted hosts may be attached to the network using an MLS SDOS acting as a front-end access machine. The access machine permits access to special-purpose machines such as parallel processors which are not likely to have direct SDOS support.

The system will evolve to permit the connection of multiple SDOS machines over an open Internet. The SDOS system will provide the necessary network protection required for the transmission of multilevel data. Untrusted single-level Cronus hosts may reside on the same network. Communications between untrusted Cronus hosts and SDOS hosts will be accomplished by using an SDOS host as a gateway.

System Overview

Figure 1 illustrates the major system components of the SDOS architecture and their relationships to each other. The SDOS user interacts with the system through the User Interface which permits him to execute a SDOS client or a user-written application client. A trusted path is provided by the system for security-critical operations performed through clients. Clients perform work for the user by issuing operation invocations to SDOS system managers or user-written application managers. The message-passing Switch on the local host is responsible for locating the appropriate manager. The Switch may need to interact with its peers on remote hosts in order to set up connections, locate the manager, route the invocation, and receive the reply. The sections to follow will describe the basic object model and major system components in greater detail.

The SDOS Object Model

The basic structuring paradigm for the SDOS system is the object model. The object model attempts to provide abstractions which closely model the way users expect to solve their problems. The model consists of objects and operations which may be performed on the object.

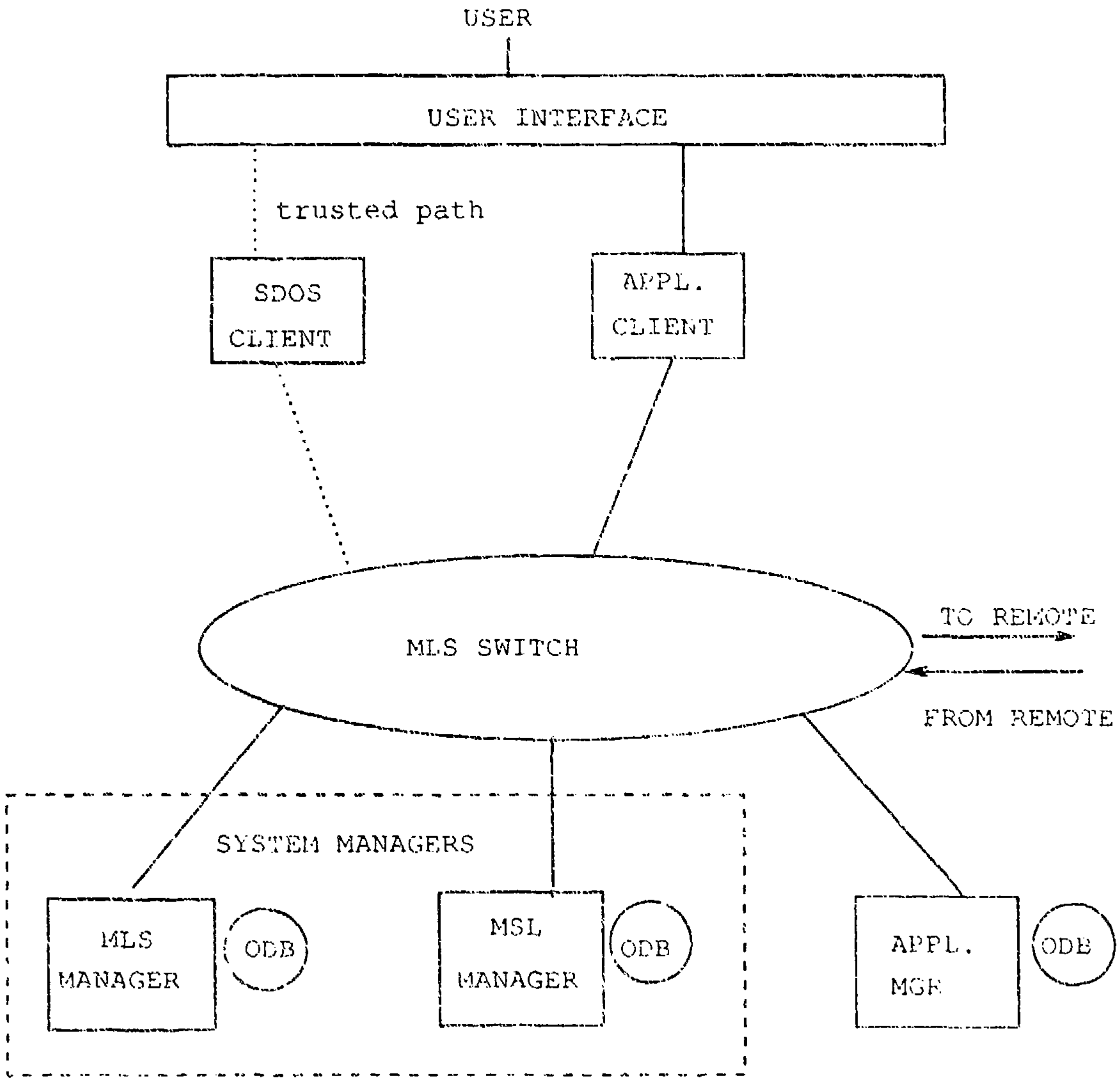


Figure 1: SDOS System Architecture

Objects are thought of as entities which satisfy certain invariants which characterize their behavior. The objects may only be accessed by a well-defined set of operations guaranteed to preserve these invariants.

Many objects may exhibit the same general behavior. It is therefore convenient to define operations on an object *type* which are valid for all objects of that type. A type is a specification of a set of objects. The public part of the specification is the operation interface which includes the operation names and operation parameters. The private part, which is not accessible to the user, includes the executable that implements the operations and the internal representation of the objects.

In general, new types may be constructed from existing types. All SDOS types form an inheritance hierarchy or Is-A hierarchy. Each type with the exception of the root type, Object, has exactly one parent. A type inherits operations from its predecessors. A type may also define new operations which are not present in its parent. SDOS provides the facility to create user-defined types using objects of type Typedef. These types must be a child of some system type or a previously defined user type.

A new type may also be defined using existing types in a Part-Of relation. The representation of the new type is composed of more primitive type definitions which may in turn be composed of even more basic types. The operations on the new type translate into operations on the more primitive types.

Basic SDOS System Types - The basic SDOS system types include:

- Host - the object associated with an SDOS host.
- Host Data and Service Data - configuration objects describing a host and its services.
- Primal Process - objects corresponding to user processes and managers.
- Session - objects corresponding to a user session.
- Principal and Group - objects associated with the identity of a user and used for authentication.
- Directory - objects used for the symbolic naming of objects.
- File - primal and fast file objects provide a distributed file system.
- Audit - audit objects.
- Typedef - objects that define an object type.

Detailed information about SDOS system types may be obtained from [SDOS 89a] and [SDOS 89b].

Object Naming - SDOS provides a global and location transparent naming facility to the user. A name is globally transparent if the name can be issued from any location and uniquely identifies an object. A name is location transparent if the location of the object is not directly encoded in the name itself. There are two levels of names for objects in SDOS. The Unique Identifier (UID) is a machine-generated internal name, and the catalog name is a user-selected symbolic name. Both are globally and location transparent.

SDOS objects have a single UID which is stored with the object and is bound to the object at object creation time. The UID is not meant to be manipulated directly by users of the system. Its internal representation is optimized for handling by the machine. The UID includes the object's type, security level, and an unique number.

Users typically want to reference objects using symbolic strings which are meaningful to them The Catalog Manager provides a distributed and replicated service which maintains the mapping between user-defined symbolic names and system-maintained UIDs. The catalog name is a hierarchical naming structure of the form ":a:b:c" where "a" and "b" are directories and "c" is a catalog entry in "b". The catalog entry is bound to the object using a specific *Create Catalog Entry* operation invoked on a directory object. Directories in this hierarchical naming structure are monotonically increasing in security level. The catalog is distributed so that different hosts may manage different parts of the name space. The upper portion of the catalog is replicated to support efficient access to different parts of the name space.

It is not required that every SDOS object have a symbolic name. An object may therefore have none, one or more symbolic names.

SDOS Objects - All SDOS objects are single-level entities. SDOS provides reliability and availability by supporting replication of objects on multiple machines. Certain objects are *primal objects* which means that they can not be replicated or migrated. Primal process objects corresponding to SDOS processes are examples of primal objects.

Objects which may be moved from host to host are called *migratory* objects. A *replicated* object is one which has been duplicated and resides on more than one host. Each replica of the object has the same UID. The object may be accessed on any of the hosts where it resides.

In response to a request to perform an operation on an object, the SDOS system must first locate a copy of the object. Since an object may be migratory, its location may vary from invocation to invocation. The SDOS system maintains the consistency of the copies for replicated objects.

The data associated with an object has a type-dependent representation. In addition to this data an object has certain attributes called *instance variables* which include its Access Control List and certain system parameters.

Clients and Managers

Clients and managers are SDOS processes. For each SDOS process there exists a Primal Process object which is managed by the Process Manager.

Managers - The implementations of types, object instances, and operations are performed in object managers. An object manager maintains the representation for all objects of a given type and implements the operations that are performed on objects of that type. It is responsible for maintaining the integrity of the object representations. A manager maintains an object database (ODB) for each type that it administers.

In SDOS, managers only exists for the types corresponding to the leaves of the type hierarchy. A manager may manage objects for one or more of these types. In general, if a manager manages more than one type, they are leaves of a common subtree. This allows the manager to implement common operations only once. However, a manager is permitted to manage types from disjoint subtrees in the hierarchy.

Single-Level and Multi-Level Managers - A manager may be a single-level object manager or an MLS object manager. A single-level object manager only manages objects at its security level and is implemented as a single-level process.

An MLS object manager is one which is able to handle operations on a type for objects at a range of security levels. An MLS manager may be implemented as a single multi level process or multiple single-level (MSL) managers. If it is implemented as a single multi level process, then the manager is part of the mandatory TCB and is trusted to perform mandatory access checks.

If the manager is implemented as a collection of single-level managers then each single-level manager manages objects at its level. It trusts that the system routes messages to the correct manager; MSL managers perform no mandatory access checks. Invocations on objects at the same level as the client are handled by single level managers at that level. Invocations up (writes) on objects that dominate the level of a client are routed to the single level manager at the object's level. No response is given to the client since this would constitute a write down. Invocations down (reads) on objects that are dominated by the client are handled by the manager at the client's level. Single-level managers can read down directly into an object database at a lower level.

In general, the functionality of any MLS manager may be implemented as a single multi-level process or as a collection of single level managers. This decision is made on a manager-by-manager basis and must take into consideration TCB minimization, system resources consumed, and performance requirements of the service. This decision may depend on hardware and software architectures of the machines the manager will run on.

System Managers and Application Managers - There are two types of managers within SDOS, the SDOS system managers and user-written application managers. System managers are registered with the SDOS system to manage one or more of the basic system types. All system managers are multi level object managers.

The basic system managers are:

- Process Manager - manages SDOS processes and session objects.
- Authentication Manager - manages authentication objects.
- Catalog Manager - manages directories and their entries.
- File Manager - manages files.
- Configuration Manager - manages host and service configuration objects.
- Host Manager - manages a host.
- Audit Manager - manages audit objects.
- Typedef Manager - manages type definitions.

A user-written application manager manages only user-defined types and is not permitted to manage SDOS system types. These managers may be single-level managers or multi-level managers constructed using a collection of single-level managers. Both types of managers are permitted because this construction does not extend the SDOS mandatory TCB. Application managers may be written using SDOS provided tools to build single-level managers.

Generic Objects - Each SDOS type has a *generic object* which is used for a variety of purposes. The generic object is associated with the manager for the type. It is referenced by a generic UID. A user may invoke an operation on the generic object by specifying the generic object's UID and the operation. These operations include operations on the type (usually referred to as *class* operations in object-oriented languages) such as object creation, or operations which reference a collection of objects such as listing all the objects of that type. The operation which locates the manager for an object is a generic operation.

DAC - All managers enforce discretionary access control on their objects. An Access Control List (ACL) is maintained for each object which indicates which users may perform which operations on that object. The ACL also maintains a list of negative entries. The DAC policy is necessarily object-dependent since operations and their semantics vary with the type. Therefore, each manager is part of the discretionary access control TCB.

Replicated Managers - The SDOS system may have one or more managers which manage the same type at the same security level. These managers must, however, reside on different hosts. These managers may be configured to manage different objects of the same type or maintain the consistency of replicated objects.

Clients - A client process is any process which acts on behalf of a user and performs work by invoking operations on objects. There are two types of clients, SDOS-provided clients and user-written application clients. SDOS clients may include trusted software which has been demonstrated to be free from Trojan Horses and can be trusted to reflect the user's intentions. These clients may be invoked through the Trusted Path. User-written application clients have no such assurances. All clients are single-level processes.

Principals and Groups

Every SDOS user or manager has a principal name which is stored in a corresponding principal object. Managers have principals names which correspond to the name of the manager. The principal object is managed by the Authentication Manager which may be replicated. Every principal object contains a list of groups to which the user belongs. When the user logs in, a default group is enabled and becomes active. There may be groups to which the user belongs that are not enabled automatically. Every group object contains a list of the principals that belong to that particular group.

SDOS principal and group names are globally transparent. They are used by managers to perform DAC checks to determine whether a user is permitted to perform the requested operation on the specific object.

The SDOS Switch

The SDOS message-passing Switch is an MLS process and is part of the mandatory TCB. The following sections will discuss the protocol used by clients and managers in communicating with the Switch and the major functional components of the Switch.

Operation Protocol - The Operation Protocol is used for communications to the Switch by clients and managers. The basic inter-process communications (IPC) primitives are:

- Invoke - invokes an operation on an object.
- Send - send a message directly to a process.
- Receive - receive the next message.

The Invoke is used to invoke an operation on an object. A manager handling an invocation may need to perform secondary invocations on other objects (possibly of different type) to complete the primary invocation. The SDOS Process Support Library (PSL) provides a higher level abstraction to the user. It consists of a set of synchronous remote procedure calls for the common operations on a type. The Send is used by managers to send a message directly to the requestor in response to an invocation. The Receive is used by both managers (to get the next invocation or response from a secondary invocation) and clients (to receive the reply to a primary invocation).

The Switch supports these IPC primitives using three basic message types that define the Operation Protocol:

- Request - used to support invocation messages.
- Reply - used to support send messages.
- Forward - used by a manager to forward an operation to another manager of its type when it determines it is more appropriate for the invocation to be performed by the second manager. This may be determined based on resource allocation considerations.

Operation Switch and Locator - The SDOS Switch is responsible for routing operations from clients to the correct object manager based on the object's UID. The Switch is composed of a *Locator* and an *Operation Switch*. The Locator determines the host location of the object. If the object is of primal type, then the invocation must be routed to a manager on the local host. If the object is not primal and if the object is not present locally, then the Locator must determine the host location of the object. The object's location may be present in a local object cache if a message was previously sent to this object. Precautions must be taken in the design of the multi-level cache so that covert channels are not introduced. If there is a miss on the cache, the Locator performs a Locate operation on the generic object of the type using the network's broadcast mechanism. All managers which have a copy of the object will respond positively to the Locate.

Once the object is located, the Operation Switch routes the operation to the Switch on the appropriate host. An Operation Switch maintains IPC connections with all local clients and managers and network connections with remote Switches. It also listens for request for new connections either locally or from remote hosts.

The Layered Implementation

SDOS is implemented using a layered architecture. The SDOS clients, managers, and Switch are implemented on top of an existing secure Constituent Operating System (COS). An important goal of the design is that SDOS be implementable without modifications to the COS and implementable on a system of heterogeneous COSs. The COS must meet TCSEC B3 security and assurance requirements. The following features of the COS are used:

- mandatory access controls
- discretionary access controls
- user and process identification and authentication
- trusted path
- local IPC

- TCP/IP and UDP to a remote host
- file system
- device support

This approach requires that SDOS labels, processes and objects be mapped onto COS labels, processes and objects. Communications between SDOS processes on the same host must use the COS's IPC mechanisms. SDOS persistent objects, like a manager's object database, must be stored in COS files. The COS's mandatory access control mechanism is relied on to enforce that only processes at the same level may communicate in a two way conversation. SDOS clients may make invocations that result in a write-up to objects that dominate them. However, no reply is returned which indicates the success or failure of this operation. SDOS makes a best-effort attempt at performing the operation. SDOS processes may also read down directly into COS files that they dominate, subject to the COS's discretionary access controls. This ability is used by the MSL implementation of managers. The COS's discretionary access control mechanisms are used to isolate what COS objects may be accessed by different SDOS principals. For example, the file which implements a manager's object database is protected from direct client access using COS DAC controls. The COS is relied on to identify and authenticate the COS user associated with every COS process. SDOS will then map the COS user into an SDOS principal. This identification is necessary for the enforcement of SDOS DAC policy by managers and for the identification and authentication of system processes such as managers to the Switch.

Clients and managers are COS processes which communicate with the Switch using the COS's IPC mechanisms. Clients and managers communicate with the Switch using the Operation Protocol implemented on top of the basic IPC mechanism. If the Switch must communicate to a remote host, it uses the COS's TCP and UDP communications facilities. The Switches communicate using the Inter-Host Protocol implemented on top of TCP and UDP. Managers use COS files to implement their object databases.

Operation Invocation Scenarios

In this section we discuss the possible operation invocation scenarios. We denote the client's security level as SL_c and the object's security level as SL_o. MLS and MSL managers have a security range for which they may handle operations request for, this is range denoted as $SL_{min,max}$, where the maximum security level dominates the minimum security level. A particular single-level manager that is part of a MSL manager has a security level denoted as SL_{msl}, where $SL_{msl} \in SL_{min,max}$. A pure single-level manager only manages objects at one level and can not read down into object databases of lower levels.

We discuss the cases when a client invokes an operation on an object at its level, when the object's level dominates the client's level, and when the client's level dominates the object's level.

Object's Level Equals Client's Level

A client invokes an operation on an object at its level, $SL_c = SL_o$. The client performs this invocation by using the Invoke primitive. The invocation results in a Request message labeled at the level of the client which is sent to the local Switch. The Switch ensures that the message is labeled correctly. From the UID of the object, the Switch is able to determine its type and the object's security level, SL_o.

Each Switch has an internal table of registered managers, their types and the ranges of security levels that they manage. If the operation is generic, the Switch needs to locate any manager of that type which is able to handle messages at SL_o. If the operation indicates a specific object, the Switch must locate a manager which has a copy of the object. If the manager is not on the local host, the local Switch looks in its object cache. The location of the object may be in this cache if a previous message was routed to it. Each entry in the cache contains a (Object UID, host ID) pair and is labeled at the level of the original invocation that created the cache entry. This prevents a lower-level Locate from using a cache entry created by a previous higher-level Locate. This would constitute a covert channel. If a cache miss occurs, the local Switch broadcasts a Locate request for the object. This request is labeled at the level of the object. A remote host will forward the Locate to a manager of that type with a range that includes the security level SL_o. A manager will respond to the Locate if it knows about the object. As a result of the Locate, the message is routed to the appropriate host and an entry is made in the local object cache. More than one host may respond with a positive confirmation to the Locate request.

The remote Switch routes the message to the appropriate manager. The manager may be an MLS manager where $SL_o \in SL_{min,max}$. If the manager is an MSL manager where $SL_o \in SL_{min,max}$, then the Switch routes it to the single-level manager where $SL_{msl} = SL_o$. Lastly, the manager may be a single level manager at SL_o. The message received by the manager indicates the client's process UID, object UID, operation to be performed, operation parameters, principal and group. The principal and group are added by the local Switch and are based on prior user authentication to the system.

The principal and group are used in the manager to determine if the operation is permitted on the specified object. The object's ACL is referenced to make this determination. After performing the operation on the object, the manager uses the Send primitive to send a Reply message directly to the client.

Object's Level Dominates Client's Level

A client invokes a operation on an object whose level dominates that of the client, $SL_o \geq SL_c$. The Switch recognizes that the operation is on an object whose security level dominates the client's. The Request message which was labeled at the level of the client is upgraded by the Switch and is now labeled at the level of the object.

The same procedure as described in the previous section is used to locate a manager for the object. Locating the manager for the object is done at the security level of the object. MLS and MSL managers with $SL_o \in SL_{min,max}$ may respond to the Locate. Single-level managers at SL_o may also respond.

The operation may or may not be performed by the manager depending on the operation requested. Only write-up operations will be performed. In any case, no reply reaches the client since its security level is dominated by that of the manager. An MLS manager is trusted not to reply at any level to the client. An MSL manager or single-level manager can not reply since any message sent to the client is mediated by the Switch which recognizes that the manager's level dominates the security level in the client's UID, and does not deliver the reply.

Client's Level Dominates Object's Level

A client invokes an operation on an object whose level is dominated by the client, $SL_c \geq SL_o$. The Request message is labeled at the level of the client. The same procedure as in the previous sections is used to locate the object. The Switch may use all cache entries dominated by the security level of the client. If no cache entry exists, a Locate operation at the level of the client is performed. MLS and MSL managers with $SL_c \in SL_{min,max}$ and $SL_o \in SL_{min,max}$ will respond to the Locate. Single-level managers at SL_c do not respond since they can not access an object at a lower level. If the manager is MLS, it can look directly into its multi-level object database to determine if the object exists. If the manager is MSL, the Locate is sent to the manager at the client's level. The manager must read down into the object database maintained by the single-level manager at the object's level to determine if the object exists. After receiving the response to the Locate, a new cache entry is created in the local object cache at the level of the client.

The operation may or may not be performed depending on the operation. If the operation involves a write down into the object, it is not performed. An MLS manager is trusted not to perform any operations which involve a write down into the object. If the manager is an MSL manager, a single-level manager at the level of the client has received the invocation. Since the object resides in an object database which is a COS file labeled at SL_o, it is prevented from performing a write operation by the COS's mandatory access control. Operations which are read downs are permitted, and a reply is returned to the client at its level.

Current Status and Future Plans

SDOS is a thirty-month project ending in early 1991 with a demonstration of the prototype system. The basic system requirements, system architecture, and security policies have been completed. The development of the formal security model and detailed software design need to be completed before implementation of the system is begun. Forthcoming papers will discuss the security policy for SDOS and detailed design issues regarding the major SDOS system components.

References

[Cronus 86] R. Schantz, R. Thomas, and G. Bono, *The Architecture of the Cronus Distributed Operating System.* Proceedings of the IEEE 6th International Conference on Distributed Computing Systems, May 1986, pp. 250-259.

[SDOS 89a] R. Wong, et al., *System Segment Specification for the SDOS.* ORA Technical Report, TR 25-1. Feb. 1989.

[SDOS 89b] R. Wong, et al., *System Segment Design Document for the SDOS.* ORA Technical Report, TR 25-2. June 1989.

[TCSEC 85] *DoD-5200.28-STD, DoD Trusted Computer System Evaluation Criteria.* December 1985.

[TNI 87] *NCSC-TG-005, DoD Trusted Network Interpretation.* July 1987.

Toward a High B Level Security Architecture for the IBM ES/3090 Processor Resource/Systems Manager™ (PR/SM™)

Thomas T. Russell
IBM Corporation
Poughkeepsie, NY 12602

Marvin Schaefer
Trusted Information Systems, Inc.
Glenwood, MD 21738

Background

For several years now, IBM has been evolving its MVS and VM System/370 Operating Systems to higher levels of security. First, in the mid-1970s and early 1980s with the implementation of system integrity and then with the implementation of the Resource Access Control Facility Product on those systems. In 1988, IBM announced its intention to evolve the security of MVS and VM by providing functions that are designed to meet the Class B1 Trust Requirements as defined in the *Trusted Computer System Evaluation Criteria (TCSEC).*

As part of this evolutionary process IBM is exploring[1] the idea of using the logic.:l partitioning facility implemented in the Processor Resource Systems Manager (PR/SM™) of the IBM ES/3090 Processor Complex to increase the risk-range of data that can be processed concurrently within a single processor complex.

Basic Concept

IBM is investigating the possibility of performing security engineering on its IBM ES/3090 PR/SM product line. It is believed the new security engineering could produce an overall system structure that would enforce a mandatory security policy with high B-level assurances and which, when coupled with appropriate evaluated

[1] **Disclaimer:** The information presented in this paper should not be viewed as a commitment by the IBM Corporation to implement changes to the Processor Resource/Systems Manager (PR/SM™) Feature of the IBM ES/3090 Processor Complex, nor is any implication intended that the National Computer Security Center has agreed to evaluate a product having an architecture such as that described in this paper.

B1/C2[+] commercial operating system products[2], could be evaluated to a high B level of a suitable interpretation of the (*TCSEC*).

The PR/SM TCB would be based on the Rushby Separation Kernel concept. PR/SM would enforce a mandatory security policy that partitions a single computer mainframe into up to 7 *completely* isolated security domains, such that no user or user process associated with any one partition element could communicate with any user, process or object associated with any other partition element. Each partition would be configured with an independently-evaluated trusted operating system. It is this operating system, evaluated at or beyond the C2 level, that enforces discretionary access control (and other required controls) on its uniquely identified users.

Separation Kernel Concept

The IBM ES/3090 Processor Resource/Systems Manager (PR/SM) provides a hardware/microcode base sufficient to support the efficient implementation of a "separation kernel." The separation kernel concept was introduced by John Rushby in 1982[3] to serve as a simple, but high-assurance, model for sharing a computing resource between completely isolated subjects.

A separation kernel is similar to a virtual machine monitor. It masks sharing of a single physical machine by simulating conceptually separate "machines". It does this by providing each user with a private, isolated, "computer" environment, complete with its own address space, set of dedicated devices, I/O channels, etc.

A separation kernel is *very* simple. Unlike some commercially-available virtual machine monitor systems, the separation kernel builds an absolute partition of the physical computer resources to which its users are assigned. Each partition element is viewed as if it were a complete computer system unto itself. Unlike a virtual machine monitor, a separation kernel does not provide services or resources beyond those available on the selected computer: there is no simulation of exotic devices, nor of memory beyond that available on the physical computer. No

[2] The B1/C2[+] concept derives from Appendix A of the *Trusted Network Interpretation of the TCSEC (TNI)*.

[3] J.M. Rushby, "Proof of Separability -- A Verification Technique for a Class of Security Kernels," *5th International Symposium on Programming*, Turin 1982

resource is directly shared between the partition elements. Hence, I/O devices would not be dynamically shared between users in different partitions.[4]

In Rushby's original exposition, each user was assigned to a unique "machine". Each machine was identified by a unique color, e.g., RED. Rushby's rules of separation were stated as follows:

- Effects perceived of operations performed on behalf of user on the RED "machine", for example, must be capable of complete description in terms of objects known to the RED "machine" (i.e., there must be no communication from any other machine to the RED machine);

- Users on other machines must perceive no effects when operations are executed on behalf of user on the RED "machine" (i.e., no sequence of actions performed on the RED machine should be perceivable from any other machine);

- Only RED I/O devices may affect state perceived by the RED user (i.e., no I/O devices, other than those configured to the RED machine, shall be able to affect the state of the RED machine);

- I/O devices must not be able to cause dissimilar behavior for states perceived by the RED user as identical;

- RED I/O devices must not perceive differences between states perceived as identical by the RED user; and

- Selection of next operation to be executed on behalf of the RED user must depend only on objects known to the RED user.

Physical isolation is sufficient to demonstrate the absence of direct storage channels between the partitions. The technical formulation of restrictions above is sufficient to demonstrate the absence of covert storage channels between the partitions.

[4] However, depending on the objects supported by the separation kernel abstraction, it may be possible for a physical environment to be configured to share a common device between two different partitions. For example, if I/O channels are the objects supported by the separation kernel and devices are only accessed via the controlled I/O channels, device-sharing could be achieved by connecting a physical device to two separate I/O channels, one from each partition.

A refinement of the separation kernel concept was proposed within the National Computer Security Center in 1984 as the foundation for Project VIKING by Bret Hartman[5]. The VIKING separation kernel would support the attachment of multiple users to each partition, but was otherwise identical to Rushby's concept. In particular, while users within a partition could create processes that could share resources and communicate with each other via an interprocess communication mechanism, there would still be no sharing or other form of communication between partition elements.

PR/SM Logical Partitioning

The Logical Partitioning Facility implemented by the PR/SM feature of the IBM ES/3090 Processor Complex is a mode of operation selected by the system operator at the time the processor complex is powered-on. Resources to be allocated to each of up to seven partitions are defined by the system operator before activation of a logical partition. After a logical partition is defined and activated, a supported operating system can be loaded into that logical partition. Central storage and expanded storage are defined to logical partitions before activation. Storage resources are allocated in one megabyte contiguous blocks (up to a maximum per complex of 512 megabytes for central storage and 2 gigabytes for expanded storage.) These allocations are static upon activation and sharing of storage resources among multiple logical partitions is not allowed. Central storage is cleared upon activation and deactivation of a logical partition. Expanded storage, which is similar to central storage, is cleared upon logical partition activation.

Individual channel paths may be allocated to each logical partition. A channel path can be allocated only to one logical partition at a time. Channel paths which have been specified as reconfigurable can be dynamically reconfigured between logical partitions if the operating systems running in the partitions support such reconfiguration. The ES/3090 Processor Complex allows for I/O sharing between logical partitions through physical channel connections only.

Central Processors (CPs) can be dedicated to a single logical partition or shared among multiple logical partitions. The allocation of CPs to a logical partition is made when the logical partition is activated. Reallocation of CP resources dedicated to a logical partition requires deactivation of the logical partition. For CP

[5] B.A. Hartman, "A Gypsy-Based Kernel," *Proc Symposium on Security and Privacy, Oakland 1984.*

resources which are shared among logical partitions, the amount of CP resource allocated to each partition may be dynamically determined or may be allocated as a fixed amount of time. Unused CP time within fixed allocations is not made available for use by other sharing partitions.

An optional vector facility that is installed on a CP is available for use by all partitions that will perform on that CP. CPs that are dedicated to a logical partition (including associated vector facilities) are available only to that logical partition.

The 3092 Processor Controller is a stand-alone support unit which initializes the system, distributes microcode to writable control storage at initialization and provides error recording, recovery and diagnostic support for the processor complex. It provides the control unit function for the attached display stations which serve as the system and service consoles.

The part of the PR/SM Feature implemented in microcode executes in a private domain that is not accessible for either read, write, or modification access by software in logical partitions.

The system console for the processor complex is used to control the operation of the ES/3090 hardware and is also used to control selection of I/O configurations, loading of operating systems in partitions, and the allocation of resources to partitions. As currently implemented, the system operator is in complete control of the system console and these important security relevant facilities. To meet the high B levels of the TCSEC, it is assumed that the PR/SM TCB would have to support separate operator and administrator functions and the rôle of a security administrator. Discussions elsewhere in this paper assume the existence of the System Operator (SOP), System Security Administrator (SSA) and System Security Officer (SSO) rôles.

Perceived Security Benefits of PR/SM

The PR/SM TCB is capable of segregating the resources on the physical 3090E mainframe into a partition, such that no machine resource is concurrently in two distinct partition elements. Each of the operating environments for using PR/SM securely is built by integrating appropriate physical, procedural and automated isolation controls. In this sense, the use of the trusted system is much like a concurrent automation of periods processing.[6]

[6] "Periods processing" is described in the Color Change section of this paper.

- **Physical Isolation Controls:** A physical processing environment is established for each security level. With the exception of the location of the physical mainframe, each environment resembles a controlled-access facility in that it contains all of its required I/O devices, its computer operator stations, its user terminals and workstations, its printers, etc. This area may be subdivided into specific areas for computer operation and user populations. As required, the area may be further isolated by TEMPEST and other protection controls. The protected environment forms the physical partition for a PR/SM environment, and is externally connected to the ES/3090 mainframe, which is protected to the level of the facility.

- **Procedural Isolation Controls:** All access by personnel to the protected processing environment is controlled by procedures. In particular, it could be required that all operations or user personnel permitted to enter the isolated facility must be identified by guards or other security personnel as being sufficiently cleared and authorized to do so. Similarly, it could be required that all device access is either obtained from within the isolated processing environment or via appropriate cryptographic isolation. System security officers would define the processing environments, selected operating systems, etc. for each of the partitions that will be controlled by the PR/SM TCB.

- **Automated Isolation Controls:** The PR/SM TCB must provide controlled access to multiple levels of classified data by dedicating and constraining the accesses of each partition to a unique security level. The PR/SM TCB would automatically create each isolated processing partition with its own defined copy of a selected B1/C2$^+$ trusted operating system.

The automated isolation controls of PR/SM and the ES/3090 are believed sufficient to provide no less assurance than would be obtained from operating Dedicated Machines in Isolated, Controlled Environments. In this sense, the trusted PR/SM system would provide concurrent periods processing for up to seven separate security levels at a time.

It appears that there is the additional potential to use an appropriate operating system evaluated to the B2/C2^{+} or higher[7] level of the *TCSEC* in the individual partitions. This would seem to provide the capability for the total PR/SM system to support a very broad span from the least cleared user to the most sensitive classified data on the system (e.g., raising the possibility of securely supporting 21 or more security classes at a time, with potentially equally large composite risk ranges), certainly more than in any traditional trusted system environment.[8]

The PR/SM TCB will be a separation kernel. This kernel will permit each of the partitions to run with its own configured operating system or special application system. Each of these systems could be configured, in concert with a system accreditation plan, to support customized access control policies within the partition.

The PR/SM TCB can also provide stronger assurances of control over what happens in each of the partitions than is guaranteed by stand-alone network components in a multilevel network. This is because the PR/SM TCB maintains continuous control over each partition and has the ability to define, constrain, create or destroy partitions precisely according to the requirements of a central system security officer. Unlike the situation in a network, there are extremely high technical assurances that any command issued by the security officer will be immediately enforced on the partition.

In many respects the PR/SM controls are superior to those provided in manual periods processing installations. The TCB provides exacting controls over concurrent periods processing applications, reducing the possibilities of operator error. In addition to being able to support several distinct security levels concurrently, there is also the possibility to provide swift automated support for additional dedicated processing environments that can be enabled sequentially when other isolated environments conclude their processing. The PR/SM TCB can provide fast assured color changes.

[7] The reason for B2/C2^{+} vice B1/C2^{+} is to provide requisite assurance for trusted labeling within the individual multilevel virtual machines.

[8] We argue that PR/SM could offer this capability precisely because of the strength of its partitioning of the physical machine into completely isolated processing environments. Unlike a traditional networking configuration, there is no potential communication provided between partition elements, and there is therefore no direct or indirect means of sharing data at a common security level between two partitions. This eliminates the potential for security concerns that would be posed by the Cascade Problem, defined in the *TNI*.

3-State Hardware

The standard IBM S/370 architecture provides two processing states. Implementations of IBM operating system products have made effective use of the two states by reserving the "supervisor" state for the operating system and other privileged programs, and confining customer applications programs in the unprivileged "problem" state.

In distinction to the IBM S/370 computer family, the IBM ES/3090 Processor series on which PR/SM is implemented provides the logical equivalent of *three* environment-specific execution states: The software running in each of the individual partitions is presented with the logical equivalent of a two-state IBM S/370 family computer; the PR/SM Separation TCB is run in its own private domain of execution. This security architecture provides the mechanism for efficiently implementing and servicing completely isolated partitions by a tamperproof TCB and support domain.

Compatibility with the IBM S/370 XA series architecture and instruction set is provided in each partition in the ES/3090. This is sufficient to permit each of the partitions to be configured[9] with a designated B1/C2$^+$ TCB that executes with the protections and privileges of the IBM S/370 "supervisor" state. Meanwhile, the non-privileged S/370 "problem" state can be reserved for all user applications programs within each partition.

Separation TCB

The PR/SM Separation TCB is implemented in hardware and microcode. Support for the System Security Officer (SSO), System Security Administrator (SSA), and System Operator (SOP) rôles will be provided at controlled workstations.

[9] "Configuration" as used here refers to the software configuration of the specific B1/C2$^+$ TCB, as opposed to a hardware configuration.

The PR/SM Separation TCB will consist of the following principal modules:

- Primitive address management
- Primitive I/O channel management
- Primitive exception management
- Cache management
- SSO/SSA support
- System operator (SOP) support
- Partition Isolation Integrity (configuration database consistency checking [Trusted Process])
- Primitive Partition Scheduling (event-driven, priorities, load balancing [optional])

Mechanisms

Much of the assurance gained from the PR/SM Separation TCB is based on its implementation in an appropriate combination of hardware, microcode and software mechanisms. The decision to place many of the mechanisms in hardware or in microcode that operates in the IBM ES/3090's unique TCB domain helps to ensure that the PR/SM Separation TCB can provide consistent and efficient low-overhead policy mediation services while providing high assurances that it can neither be bypassed nor illicitly modified.

The PR/SM Separation TCB principal functions will be implemented with the mechanisms identified below:

PR/SM Separation Kernel: Implemented in a combination of hardware and microcode.

Partition Configuration Databases: These databases are the internal representation used by PR/SM to define the isolated single-level partitions.

Environments: The environments constitute the single level areas in which operational personnel, users, terminals, operator consoles, data devices, etc. are confined. The implementation mechanism is primarily one of physical partitioning, supplemented by the isolation of I/O channels and subchannels by the PR/SM Separation TCB.

Partition TCBs: Each partition will operate at a single security level. Single-level partition-high policy enforcement will be provided by a selected B1/C2' TCB. This TCB will be defined by the SSO/SSA at the time the particular partition is defined and the TCB will be IPLed into the partition by the SOP whenever the partition is to be enabled. The SOP will IPL the configured B1/C2' product prior to passing control to the partition.

Policy

The formal access control policy to be implemented by PR/SM is enforced both by the PR/SM Separation TCB and by a B1/C2' TCB in each partition. The PR/SM Separation TCB isolates defined partitions from one another and enforces the mandatory isolation policy. The B1/C2' TCBs, executing in the individual partitions, enforce discretionary access control and perform other partition-specific services, e.g., audit, identification and authentication, etc.

Separation TCB Policy:

From the perspective of the PR/SM Separation TCB, the **Subjects** are the SSO, SSA, SOP, and the individual single-level Partitions. At this level of abstraction, the **Objects** are channels, devices, and address domains. The separation policy permits the SSO and SSA to define partitions, the SOP to enable or disable partitions, and confines each partition to those resources and operations defined to the partition (i.e., it prohibits any one partition from communicating with or referencing any other partition).

The **Mandatory** access control policy is *full* isolation by formal security level (classification and category set) for up to seven active Mandatory Security Levels at a time; the **discretionary** access control policy applies to the SSA/SSO, who define only Devices and Channels authorized to the partition, and to the SSA/SSO/SOP where ID and authorization profiles, real machine resources (e.g., CPs) are applied.

B1/C2$^+$ TCB Policy

Within the B1/C2$^+$ TCBs, the Subjects are defined as Users, process in execution, environment-specific operators and administrators; and the Objects are Files, Data Sets, Inter-process communications, cross-memory services, Devices, Terminals, Local [within environment] Machine Resources, etc.

The **Discretionary** access control policy enforced within the B1/C2$^+$ TCBs provides individual-controlled assignment for application-defined named objects; basis is per named user or group or NACL or Negative group, by specific mode of access.

Policy Implementation Mechanism

Partition

PR/SM's **Mandatory** access control policy is enforced by the PR/SM Separation TCB, which implements the completely disjoint single-level partitions. Each partition is accorded physical and logical access to precisely those devices that have been physically placed within the partition's environment. The partition's devices and I/O channels will all be treated as though they are classified at the security level of the partition. The logical assignment of these devices and I/O channels to the partition represents a **Discretionary** access control decision made by the SSO/SSA at the time the partition is defined. These discretionary configuration definitions can be redefined by the SSO/SSA, but may be enabled only after a Power On Reset is performed if channels need to be reallocated.

Object Reuse

The PR/SM Separation TCB must guarantee that all object reuse and security audit requirements are satisfied with respect to such redefinitions or reallocations of resources.

Environment

Within a particular environment, the **Mandatory** access control policy is enforced by physical isolation (although a B1$^+$ TCB may be used to provide security labeling within the partition).

Enforcement of the **Discretionary** access control policy within the single-level environment will be performed by designated NCSC-evaluated B1 Trusted Operating System Products that exhibit the *TNI*-required enhanced C2$^+$ Policy and Audit Features.

SSO/SSA and SOP Functions

The functions of the SSO, SSA and SOP must be controlled directly by the PR/SM Separation TCB. In their official interactions with PR/SM, these personnel have only limited rôle-based capabilities, according them limited special-purpose functionality from authenticated isolated domains. The Separation TCB will require positive operator login to provide an identification function and enforce individual accountability for all actions, based on the official's identity and rôle.

The SSO, SSA and SOP can only be permitted to perform their functions from terminal positions defined by the PR/SM TCB. The privileges accorded these personnel may differ according to individual identity, so that, e.g., not all SSAs or SSO should necessarily be able to perform the same acts.

To ensure consistency and simplicity, all SSA/SSO and SOP functions could be performed from menu-driven workstations.

The PR/SM Separation TCB must perform necessary legality and consistency checking on all SSA/SSO and SOP interactions, and all of these actions must be audited at the system level.

Color Change

It may be necessary to time share the IBM ES/3090 over more partitions (environments) than the maximum number that can be supported by PR/SM at any one time. Bringing one partition down and replacing it with another partition is called a "Color Change".

The ability to perform a color change will be controlled by the SSA. In order to eliminate potential covert channels, the PR/SM Separation TCB will ensure that color changes are transparent to other environments. In operation, the color change can only be achieved via special procedures required of the SOP and the SSO.

The SOP must first perform a Partition Disable action. This absolutely disables and destroys *all* partition activity, disables *all* partition channels and devices, and purges *all* partition cache and memory objects. In effect, the Partition Disable action quickly and surely renders the environment inoperative by taking away the computer resource it had been using.

The SOP then causes the creation of a new partition (and the defined environment) by invoking its *name*. The new partition cannot be enabled unless sufficient PR/SM resource is available, including defined consecutive memory locations, dedicated central processors (CPs), configured devices, etc.

Once the new partition has been created, the SOP performs an auditable Enable/Activate action in which an IPL is performed of the B1/C2$^+$ TCB that had been designated by the SSA. This IPL is performed from a controlled system image that resides on a device placed under control of the Environment's SSO/SSA. The PR/SM Separation TCB must audit the activation of the new environment. This final step also causes the connection and enabling of all of the environment's channels/devices consistent with the configuration-controlled definition.

INITIAL APPROACH FOR A TRW SECURE COMMUNICATIONS PROCESSOR

JUNE 22, 1989

BONNIE P. DANNER
TRW, INC.
2701 PROSPERITY AVENUE
FAIRFAX, VA. 22031
(703) 876-8760 OR 876-8160

Abstract

This report presents the preliminary results of a TRW investigation to define security policy objectives, security requirements, and an initial architecture for a general-purpose, multilevel secure communications processor based on an existing TRW product, the Remote Communications Processor (RCP). A goal of this TRW Internal Research and Development (IR&D) project was to lay the groundwork for a trusted (B2 level) TRW Secure Communications Processor (TSCP) in cooperation with a concurrent TRW IR&D investigating the issues for the development of a transportable, POSIX-compliant RCP. The TSCP trusted computing base (TCB) is structured in terms of an operating system kernel, trusted system functions, and trusted application functions that will collectively satisfy the derived security policy objective. The trusted communications processor initially defined by this IR&D project will be applicable to specific operational environments with MLS accreditation requirements. This study provides an initial security "road map" for a B2-level, general-purpose, transportable communications processor.

1. Introduction

This report describes the main results of a TRW security engineering effort to define an initial architecture for a multilevel secure (MLS) communications processor that will satisfy the criteria for a B2-level of trust in the Trusted Computer System Evaluation Criteria (TCSEC), DOD 5200.28-STD. Based on a successful, fielded communications product, the Remote Communications Processor (RCP) and on experience in a variety of multilevel security programs, TRW initiated an Internal Research and Development (IR&D) project to define an approach for a MLS RCP in response to the growing need in the military and intelligence communities for such a product. The RCP is an existing front end communications processor that was developed on a VAX/VMS base by TRW in response to message processing requirements for Navy Shorebased communications. The TRW experience in Navy Command and Control System communications includes the development of a major Navy upgrade, The Ocean Surveillance Information System (OSIS) Baseline Upgrade (OBU). Within the OBU project, TRW developed a multilevel secure message handling system.

Concurrent with the TSCP IR&D project, a separate IR&D project was launched to examine the feasibility of building a transportable RCP in accordance with the POSIX standards and to define the requirements for a transportable RCP. The requirements for a trusted and transportable RCP were examined cooperatively to derive a foundation for a transportable TRW secure communications processor (TSCP).

The proposed TSCP will provide communications services in the form of on-line, automated, near real time message switching for the timely receipt, display and correct routing of message traffic to a variety of destinations. The TSCP will be a front end processor hostable on computer systems that support POSIX and X Windows interfaces and will provide appropriate modem devices as required for specific environments.

1.1 Motivation for a TSCP

There is increasing need for general-purpose, multilevel secure (MLS) communications processors that meet military and intelligence community requirements for enhanced capabilities with improved security. Most current classified systems operate in system high or dedicated modes with reliance for security placed largely on procedural or physical controls. To open existing computer systems to users with differing clearances requiring access to multilevel data would pose unacceptable risks. Enhanced computing capabilities, increased operational and communications requirements, and advancements in hardware are driving the requirements for MLS.

State-of-the-art technology currently provides limited MLS operating system products, and promising new MLS products are being developed. Independent research projects and government research programs are striving to address the critical need for MLS. Some examples are: the Multinet Gateway Program, (RADC, Ford Aerospace); TRUMMP, (Magnavox); LOCK, (Honeywell); ASD, Sybase Secure Sequel Server and ASOS, (TRW); and numerous MLS operating system products. However, no general-purpose, certified B2-level communications processor is yet available. This TRW IR&D project responded to the need for a defined approach and initial set of requirements to achieve a B2-level, general-purpose secure communications processor.

While guidance exists in the TCSEC, the Trusted Network Interpretation (TNI) and numerous service and agency documents, there is no well-defined, single approach to securing specific applications. Application and interpretation of the guidance documents will be necessary to develop MLS products that meet specific defense communication needs.

1.2 Objectives and Approach

The overall objectives of the TRW SCP IR&D project were to:

- Formulate initial B2 security policy objectives and high level security requirements for the TSCP.
- Derive an initial TSCP architecture to meet B2 requirements and also address applications portability (POSIX standards).
- Define the basic steps to design and develop the trusted and transportable SCP in cooperation with the TRW Power Projection IR&D project.

The TRW approach was to redefine the architecture for a fielded, successful RCP to meet MLS requirements. Examining both B2 trust and portability issues, TRW investigated relevant, current research projects, defined requirements, and examined alternativ approaches for the SCP.

2. Initial IR&D Results

This IR&D project accomplished its initial objectives to lay the ground work for the development of a TSCP. Difficult issues will need to be solved to develop a B2 TSCP: B2 certification challenges, limitations of current technology, potential unavailability of suitable trusted products, integration of trusted applications with an evaluated product, portability goals, inadequacy of current models for application systems and distributed system complexity.

2.1 Results Overview

To define security policy objectives and derive draft security requirements, TRW analyzed issues such as interpretation of MLS guidance documents, determination of trusted application functions, identification of the trusted computing base (TCB), implementation of access controls and accountability, and the formulation of an architecture that simultaneously meets performance, trust and portability requirements. A survey of pertinent MLS research projects confirmed the need for B2-level communications processing and identified a variety of ongoing efforts responding to the technical challenges.

TRW analyzed the B2 criteria with respect to TSCP operational needs, defined security constraints and feasibility. The TSCP TCB was structured in terms of an operating system kernel, trusted system functions and trusted application functions that collectively satisfy security policy objectives.

A MLS operating system will support TSCP security requirements, and security functions will be assigned accordingly. There is risk associated with the assumption that a secure product will be available for TSCP development. However, the probability that a suitable B1-level, UNIX-based operating system will be available in 1989 is high. A multilevel secure TSCP with the required B2 supporting features can be developed on a B1 base and transported to a B2 operating system once it becomes available. AT&T has predicted its B2-level, POSIX-compliant UNIX operating system will be available by mid 1990. Other vendors are claiming they will achieve comparable products in the near future.

On another project, TRW is developing a MLS product to meet the highest achievable level of trust, A1, with an Ada-based operating system for real time applications, the Army Secure Operating System (ASOS). As one possible alternative, the SCP security architecture can be tailored to accommodate application on ASOS with a suitable operating system interface designed to meet POSIX requirements.

The TSCP security policy objective addresses a general purpose classified communications environment tasked with processing multilevel data. The TSCP must examine message traffic flowing into and out of the system, reject any messages failing to meet established security criteria, log all messages (rejected messages with reason for rejection) and export only valid messages in accordance with the security access rules for classified data, port, and destination security levels.

The TSCP must separate functions that implement security to minimize the possibility of undesirable side effects and enhance the credibility and correctness of the system. Additionally, the partitioning of the TSCP into trusted and untrusted elements, supported by physical separation and modular design will enhance the security assurance activities. The TCB will be modeled, specified, and validated in accordance with TCSEC B2-level criteria.

2.2 TCB Overview

TRW defined a trusted computing base for the TSCP that will make use of MLS operating system advances and employ a hierarchical, layered approach to trusted system design. The TSCP will provide front end, automated message switching functions for the receipt, display and routing of military message traffic. The TSCP TCB will consist of computer software, firmware and hardware jointly enforcing TSCP security policy.

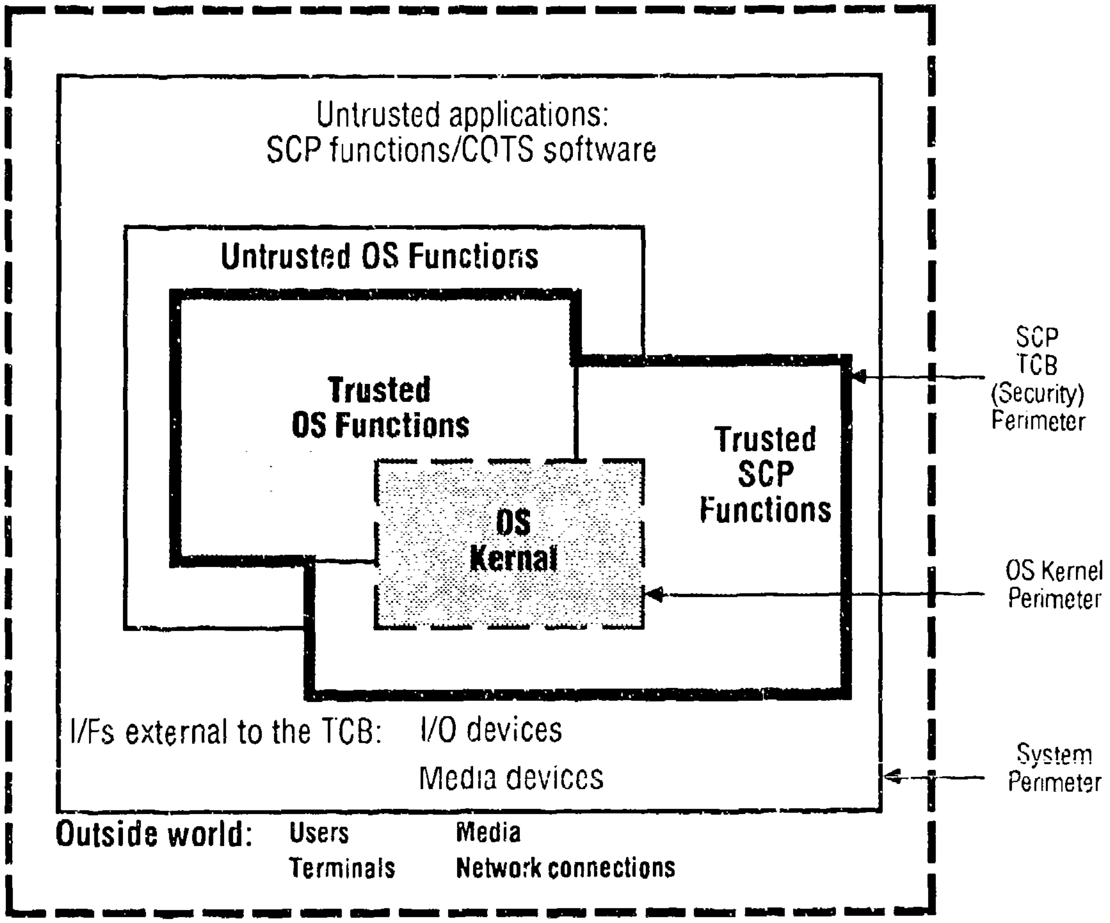
All functions of the TSCP system will be partitioned into trusted and untrusted elements. The operating system will act as a base on which the TSCP functions operate with a trusted operating system kernel mediating TSCP accesses to sensitive system objects in accordance with B2 security policy. Figure 1. presents a conceptual view of the software security layers in the TSCP.

The software portion of the TCB for the TSCP will be composed of the trusted software functions that support the security policy enforced by the B2 operating system and its TCB as well as the trusted application functions that support secure message processing, message export, message import, operator/administrator window interfaces, and security audit/alert management. Trusted functions are those that must be relied on to correctly enforce the defined security policy.

External interfaces of the TSCP will be controlled by the TCB functions for MLS processing. The security perimeter of the TSCP necessarily will include the system and application functions responsible for message flow through the TCB. The external view of the TSCP, the trusted partition, major functions and major interfaces are illustrated in Figure 2.

TSCP TCB functions will include:

- review of all communication between users and subscribers
- identification and authentication of system users
- mediation of all accesses of defined objects by defined subjects where subjects are TSCP system users and processes acting on their behalf; objects are messages, sensitive files, devices and processes
- identification and maintenance of message sensitivity
- validation and correct routing of messages
- rejection of all messages that fail defined criteria
- creation and protection of security audit trail
- approval of all outgoing messages for validity and security prior to export.

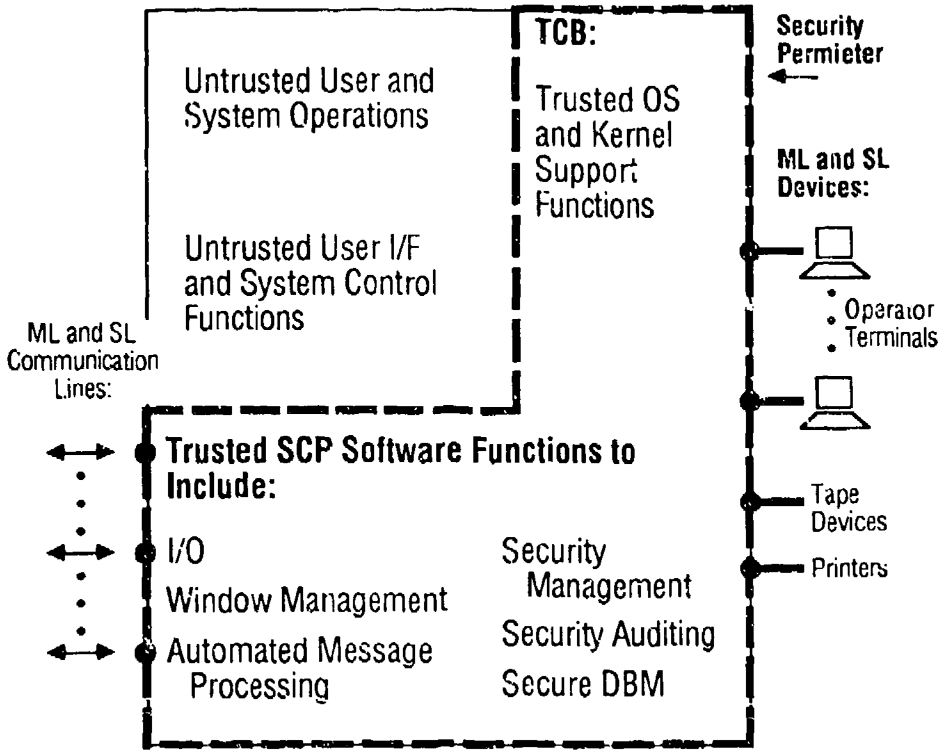


The security layers of the SCP from most critical to untrusted are:

- Operating system (OS) kernel (most critical)
- Other trusted OS functions
- Trusted SCP: Kernel I/F
  Trusted Applications
- Untrusted Applications

Figure 1. SCP Software Security Layers

External View of the SCP TCB



Untrusted User and System Operations

Untrusted User I/F and System Control Functions

TCB:

Trusted OS and Kernel Support Functions

Security Permieter

ML and SL Devices:

Operaior Terminals

Tape Devices

Printers

ML and SL Communication Lines:

Trusted SCP Software Functions to Include:

I/O

Window Management

Automated Message Processing

Security Management

Security Auditing

Secure DBM

The SCP security perimeter provides a trusted boundary for system communications. All message and user interfaces are controlled by TCB functions.

Based on RCP, I/F support includes:

- Bidirectional Asychronous links (JANAP 128 Msgs)
- RIXT/SRT (Navy/AUTODIN)
- Slow Walker
- Protocol Interface Processor (PIP)
- x.25 (future)
- HLDC (future)
- ADCCP (future)

Figure 2. External view of the SCP TCB.

The TCB functions must satisfy a defined security policy that meets the criteria for a B2-level of trust.

2.3 Security Policy Objective Overview

The security policy objective represents the first step to define the TSCP approach and provide a foundation for security that is applicable to the MLS system design. The TSCP Security Policy is intended to be flexibly tailored to specific message processing environments.

The initial security policy objective:

- has general purpose applications
- defines a general message processing environment
- satisfies TCSEC B2 policy guidelines
- addresses administrative and procedural issues
- defines basic security assumptions for the TSCP environment and personnel
- defines top-level security requirements that are system enforceable:
  - --discretionary access controls
  - --mandatory access controls
  - --object reuse
  - --human readable labels
  - --message handling
  - --accountability
  - --secure data base management
  - --system integrity
- applies to the TRW Power Projection research project for a trusted and transportable communications processor
- provides a basis for the TSCP security model.

The policy objective will be applicable to a general military/ intelligence communications processing environment operating in multilevel or compartmented mode. The TSCP must also satisfy security policy for specific environments in accordance with DOD, agency, intelligence and service policies as applicable.

The policy describes a classified message processing environment capable of handling multilevel and single-level communications lines and devices and serving users cleared at different security levels, compartments and need-to-know for data access. Data confidentiality is provided at the B2 level of trust. Integrity and assured service issues are addressed as important specific requirements and design concerns to be specified outside of the security policy model.

The TSCP policy description provides a generic environmental overview and includes security policy statements that are procedural, administrative, and system enforceable. Fundamental policy statements enforceable by the computer system represent high-level security model assertions that provide a foundation for the Formal Security Model. The chosen design ultimately will determine the means of enforcement while B2 criteria mandate certain system security functions that must be implemented by the software and/or hardware of the computer system.

2.4 TSCP Architecture

The TSCP architecture defines major software functions and their security relevance for B2 policy. The software architecture can be developed on a variety of hardware systems. The current approach assumes a single system host compatible with a chosen MLS, POSIX-compliant operating system, or as an alternative, a MLS operating system combined with a B2 trusted, POSIX-compliant interface.

The operating system will be a commercial product (or TRW's ASOS) that provides a trusted base for TSCP security. The most privileged security layers will reside in operating system kernel functions. Additional trusted operating system functions will operate in less privileged layers. The security kernel will mediate subject to object access as defined by the TSCP application. The trusted kernel ensures that discretionary and mandatory access control policies are enforced.

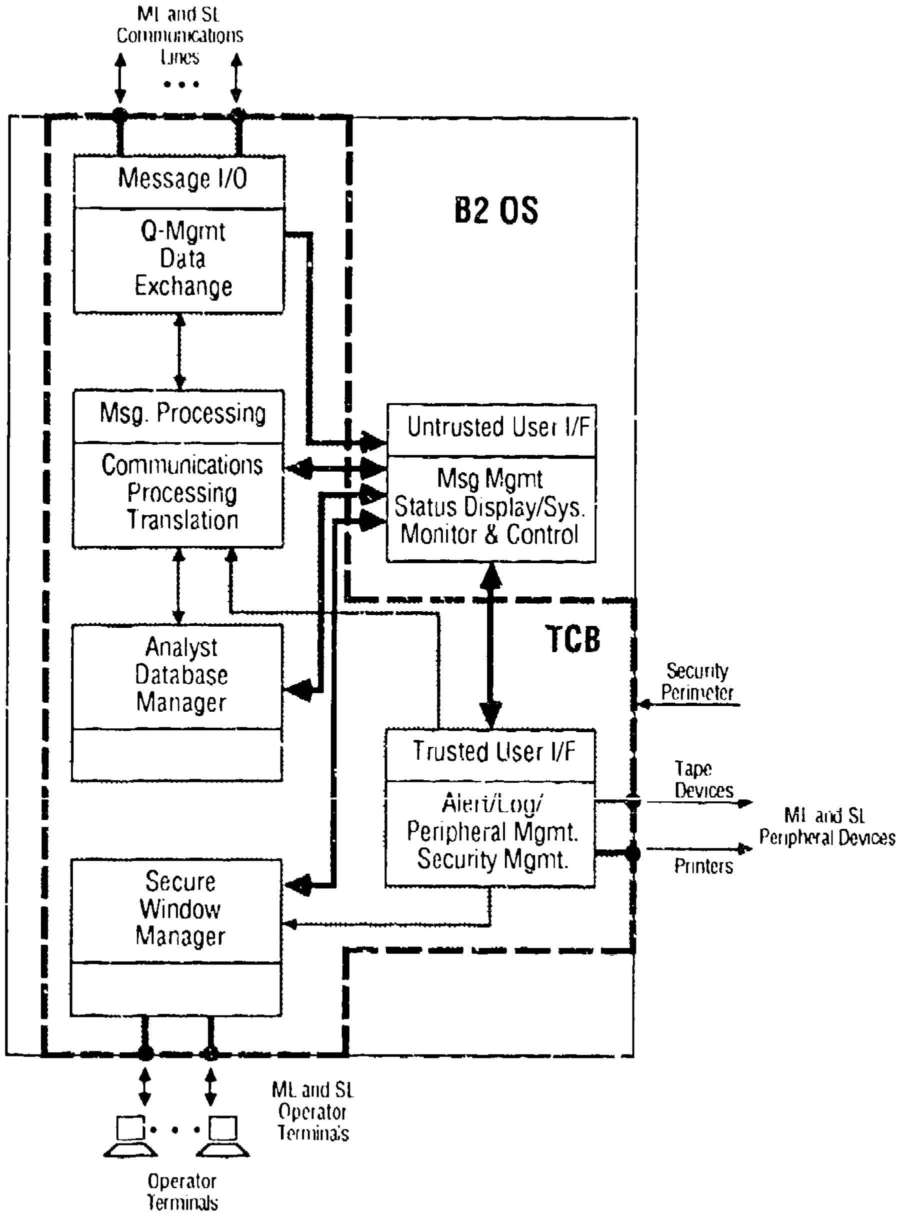
The TSCP will extend the existing operating system security policy. Specifically, the TSCP application will be a trusted software layer on top of the trusted operating system. It will include a hierarchical upper layer of the TCB and enforce security policy cooperatively with the secure operating system. Therefore, TSCP software must be evaluated to ensure that the extended policy correctly adheres to the operating system policy, and that together they satisfy the TSCP policy objectives.

The resulting TSCP security evaluation will depend on the rating of the operating system base and can achieve the operating system trust level as a maximum.

Figure-3 illustrates the major application software elements, their interfaces and the operating system base for the TSCP security architecture.

Major software elements proposed for the TSCP are:

1) Untrusted User Interface--Supports non-security-relevant user functions for message management, status displays and routine system operation and management controls. Communicates indirectly with users through the TCB Secure Window Manager, Trusted User Interface and operating system functions. The interfaces to the TCB include:
   Untrusted User Interface to
      -Secure Window Manager for display updates for operations monitoring and message management
      -Message Processing for initialization of communications data
      -Data Base Manager for message transactions


ML and SL Communications Lines
Message I/O
Q-Mgmt Data Exchange
B2 OS
Msg. Processing
Communications Processing Translation
Untrusted User I/F
Msg Mgmt Status Display/Sys. Monitor & Control
Analyst Database Manager
TCB
Security Perimeter
Trusted User I/F
Alert/Log/ Peripheral Mgmt. Security Mgmt.
Tape Devices
Printers
ML and SL Peripheral Devices
Secure Window Manager
ML and SL Operator Terminals
Operator Terminals

The TRW B2-level SCP security architecture ensures MLS enforcement through modular, trusted application processes interacting with the operating system mechanisms to satisfy a common B2 security policy objective.

Figure 3. SCP Architecture for Software Elements.

-Trusted User Interface for status data requests, message and operational alert selections and command log handling.

2) Trusted User Interface--Provides security-relevant functions for system management, security alert/log management and peripheral device management to support sensitivity labels, accountability and access control requirements. Supports security manager functions for definition and maintenance of security data files and user accountability. Relies on operating system security mechanisms to ensure policy implementation and supports trusted external interfaces for classified data export to multilevel and single-level devices. Internal and TCB interfaces include:

Trusted User Interface to

-Untrusted User Interface for current line (display) classifications, inputs to security alert/audits, display requests, statistical display,

-Secure Window Manager for display updates and valid operator modes

-Message Processing to provide communication line security

-Peripheral Devices to provide labeled external outputs.

3) Secure Window Manager--Supports secure external interfaces with TSCP system users and provides trusted labeling of window displays in accordance with MAC and DAC policies. Requires trusted operating system mechanisms to enforce security controls and interfaces with untrusted user functions to support command authorizations for identified users according to their clearances and roles. Internal and TCB interfaces include:

Secure Window Manager to

-Untrusted User Interface for commands and mode selections and valid mode requests

-External operator terminals to support labeled window displays for system operators.

4) Data Base Manager--Supports trusted data base management of sensitive analyst data for the TSCP enforcement of access control policies. Provides a trusted application layer to ensure security policy is enforced for analyst data base functions. Requires trusted operating system mechanisms for secure implementation. Interfaces with Secure Window Manager, Trusted User Interface, and Message Processing to support trusted user operations. Internal and TCB interfaces include:

Data Base Manager to

-Untrusted User Interface to send message and directory data, security log and alert entries, communications parameters, and communications data.

-Secure Window Manager to provide message displays and appropriate classifications

-Message Processing to send messages for transmission.

5) Message Processing--Provides message-level, trusted communications management. Parses, validates, translates, rejects, sanitizes, routes and transfers classified messages. Supports integrity of labels, messages and their correct association. Helps ensure secure message flow with the support of trusted operating system mechanisms to conduct communications handling and ensure the security of the message processing applications. Implements levels 3-7 of the OSI Reference Model. Internal and TCB interfaces include:

Message Processing to

- -Untrusted User Interface to provide current communications parameters and line security, for communications and status data, and to send security alerts/audits
- -Data Base Manager to provide invalid messages and and messages requiring operator attention
- -Message I/O to send messages for transmission and link control.

6) Message Input and Output (I/O). Message Input supports trusted receipt, protocol handling, collection, identification, sanitization of messages whose classification dominates security level of TSCP node and supports message information logging and acknowledgments. Message Output provides trusted message partitioning, protocol handling, checking for secure export, and secure transmission of classified messages. Message I/O interfaces with the trusted user interface to support security alerts and interfaces with message processing for message handling functions. Supports the trusted external interface with a variety of multilevel and single-level communications lines and network interfaces. Implements layers 1-2 of the OSI Reference Model. The trusted operating system provides essential support mechanisms to ensure that multilevel secure message handling policy is enforced. Internal and TCB interfaces include:

Message I/O to

- -Untrusted User Interface for message logging and management of security and link alerts
- -Message Processing for transfer of imported messages and link status
- -External Communications for import and export of message blocks, acknowledgments, and modem responses.

7) Operating System--The B2-level operating system consists of its own TCB as well as untrusted system functions. Provides a software layer that protects hardware and sensitive system elements from direct application access. The kernel contains the most privileged system functions and provides a reference monitor to mediate subject to object accesses and enforce security policy. The operating system provides system integrity, process separation, object reuse, trusted path, user authentication and identification, and secure

interprocessor communication in accordance with its security policy and in support of TSCP policy objectives. Also ensures security database access protection. The operating system interfaces with every TSCP element to support all TSCP software functions.

The full extent of the trusted software functions compared to the overall system will need to be further evaluated. The total size of the software within the TSCP system is assumed to be relatively small.

2.5 Requirements Overview

The security analysis from this IR&D project supports the first step toward the development of a prototype TSCP to meet the B2-level of trust and the POSIX standard. The initial security requirements are summarized here. The TSCP requirements will continue to be defined, and it is expected that they will be included in a system specification for a future TRW IR&D project.

An overview of the TCSEC B2 requirements is presented in Table-1. Security requirements for the first two TCSEC areas are addressed in more detail in the TSCP Security policy objective, and they were initially applied to specific TSCP elements in the Power Projection IR&D project.

The informal policy objectives were partially derived from the B2 criteria which state that the TCB for the TSCP shall be based on a clearly defined and documented formal security policy model that provides discretionary and mandatory access controls (DAC and MAC) for all subjects and objects in the computer system. An English language statement of security policy is required to define the protection requirements for the system in terms of MAC, DAC and marking policy. Message screening for security, message input and output security handling, and security auditing will be included. Specific security policies will be specified with respect to acceptable flow rules.

The security policy statements must then be written in a formal, mathematical language which presents an unambiguous description of TSCP security policy. The formal security model can be written in any mathematically-based language. The Gypsy Verification Environment (GVE) tools are available at TRW, and TRW has used Gypsy in a number of projects.

Accountability must be assured to the granularity of a single user. To ensure the security of classified information, individual accountability will be required for all access mediations of objects by subjects as defined for the TSCP.

The TSCP will require unique identification from system users and will provide trusted and protected authentication data to recognize and authorize users for specific access permissions. In addition, the system will have the capability to associate users with the

Security Policy:

- Discretionary access controls
- Object reuse
- Labels
- Label integrity
- Exportation of labeled information
- Exportation of multilevel devices
- Exportation of single level devices
- Labeling human readable output
- Mandatory access controls
- Subject sensitivity levels
- Device labels

Accountability:

- Identification and authentication
- Audit
- Trusted path

Assurance:

- System architecture
- System integrity
- Security testing
- Design specs and verification
- Covert channel analysis
- Trusted facility management

Documentation:

- Security features users guide
- Trusted facility manual
- Test documentation
- Design documentation

Table 1. The B2-Level Criteria from the TCSEC

auditable, security-relevant events they perform. The system must recognize unique message identifiers to properly handle access mediation and auditing requirements for secure message processing.

Their will be a trusted communications path between each system user and the TSCP TCB. The TSCP TCB will ensure that only the identified user can initiate communications via the associated trusted path.

The TCB will provide the capability to audit security-relevant events. Auditing includes the recording of authentication/identification activities, specified access meditations, system/security administrator actions, object deletions and transfers, rejected messages, and security alerts. Pertinent security information will be identified with each audited event as specified in detailed auditing requirements for the TSCP. An audit file will be maintained and protected by the TCB to prevent unauthorized access to security log information.

2.5 Assurance Requirements Overview

Assurance requirements for a B2-level TSCP mandate significant operational system functions and life-cycle activities. Security features that provide system protections within the software and hardware of the TSCP will be necessary. The TCB will be designed and built to provide MLS protection. Assurance activities will provide a high degree of confidence that the security requirements are correctly designed and implemented in the TSCP.

The security architecture for a B2 system is an important driver for the total system architecture. As required by the TCSEC, the TSCP will consist of a TCB which maintains an execution domain that is adequately protected against external interference. Distinct address space under TCB control will be required to maintain process isolation. Modular software functions and hardware isolation will be needed to support the protection of critical elements. Additionally, the TSCP will require analysis of potential covert storage channels and require system separation of operator and administrator actions.

Security assurance methodology for the TSCP will require a Descriptive Top Level Specification (DTLS) which provides a complete description of the TCB and its interfaces. The DTLS must satisfy the TSCP Formal Security Policy Model. Testing requirements will include testing to determine the TCB is relatively resistant to penetration and consistent with system documentation, specifically the DTLS. All discovered security flaws must be corrected. Configuration management activities will be required for the strict control of all security baselines and accurate traceability. Automated tools will be defined and used to ensure that TCB code baselines are maintained. Additional security documentation will be required: TCB design documentation (formal model and consistency evidence, DTLS, covert channel analysis results); security test plans, procedures and results; and a Security Features User's Guide and Trusted Facility Manual.

2.6 Development Methodology Requirements Overview

Trusted system development for the TSCP will require careful software engineering practices to ensure correctness. To achieve a feasible B2-level operational system, security will be integrated into the overall development process. Security is an important initial driver of the system. Basic security engineering principles apply to any good systems engineering effort. The difference will be the emphasis on satisfying security requirements.

Good security engineering practices to be applied may include: deliberate, systematic design using both hierarchical and horizontal system views; top down analysis and traceability; and use, as feasible, of a type-safe, higher order language. The TSCP will use an object-oriented design approach with an implementation language selected to meet portability and trust goals (eg., Objective C).

Security assurance requires increased analysis and review of TCB design and code, testing of security functions, testing for the overall satisfaction of the security policy, and testing for resistance to penetration. In addition, risk analysis and accreditation planning are required for an operational system for a specific application. Security risks must be carefully monitored and reduced to acceptable levels throughout the development of an operational system to achieve accreditation. Once operational, the system will require continued security assurance activities for maintenance and re-accreditation as necessary.

3. Conclusions

In this TSCP IR&D project, TRW expanded its current MLS work to explore a secure communications solution, specifically for DoD networks (e.g., DDN, AUTODIN). Recommended next steps include: a detailed analysis of alternatives, development of a formal security policy model, initiation of a dialogue with NCSC and a detailed evaluation plan, and coordination with a chosen MLS operating system vendor.

Not only are common protocols required for information exchange, but common approaches to security are necessary for interoperable MLS components. The TSCP security policy, tailored for an operational environment must be coordinated with other MLS components for a cooperative B2 policy that satisfies operational risks as a whole. This project represents a first step toward one piece of the challenging puzzle facing government, industry and academia for MLS communications.

REFERENCES

[1] D. Bell and L. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR 2997, The MITRE Corporation, Bedford, MA, July 1975.

[2] M. Carsin, R. Chapman, W. Jiang, J. Liang and D. Yakov,

"From B2 to CMU: Building a Compartmental Mode Workstation on a Secure Xenix*Base," in Proceedings of the AIAA/ASIS/IEEE Third Aerospace Computer Security Conference, December 1987.

[3] A. Cincotta, Initial Draft POSIX System Administration FIPS, NIST, September 1988.

[4] D. Clard and W. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987

[5] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December 1985.

[6] G. Dinolt, J. Freeman and R. Neely, "An Internet System Security Policy and Formal Model," in Proceedings of the 11th National Computer Security Conference, October 1988.

[7] M. Gasser, Building a Secure Computer System, New York: Van Nostrand Reinhold Company, 1988, Part III.

[8] G. King, Bill Smith, "ANFOSEC IRAD at Magnovax: The Trusted Military Message Processor (TRUMMP) and the Military Message Embedded Executive (MEX)," in Proceedings of the 11th National Computer Security Conference, October 1988.

[9] D. Kuhn, "Briefings on X WINDOW System FIPS", NIST, September 1988.

[10] R. Martin, "Briefings on POSIX FIPS 151", NIST/NCTL, September 1988.

[11] J. McLean, "Reasoning About Security Models," in Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987.

[12] J. McLean, C. Lunwehr, and C. Hertmeyer, "A Formal Statement of the MMS Security Model," in Proceedings of the 1984 Symposium on Security and Privacy, April 1984.

[13] S. Migues, "A Guide to Effective Risk Management," in Proceeding of the AIAA/ASIS/IEEE Third Aerospace Computer Security Conference, December 1987.

[14] National Computer Security Center, "Trusted Network Interpretation," NCSC-TG-005, 31 July 1987.

[15] P. Rougeau, "Integrating Security into a total Systems Architecture," in Proceeding of the AIAA/ASIS/IEEE Third Aerospace Computer Security Conference, December 1987.

[16] M. Shafter, G. Walsh, "LOCK/IX: On Implementing UNIX on the LOCK TCB," in Proceedings of the 11th National Computer Security Conference, October 1988.

[17] W. Shockley and R. Schell, "TCB Subsets for Incremental Evaluation," in Proceedings of the AIAA/ASIS/IEEE Third Aerospace Computer Security Conference, December 1987.

TRACK B

PRIVACY FOR DARPA-INTERNET MAIL

John Linn
Secure Systems
Digital Equipment Corporation, BXB1-2/D04
85 Swanson Road
Boxborough, Massachusetts 01719-1326

Stephen T. Kent
BBN Communications Corporation
150 CambridgePark Drive
Cambridge, Massachusetts 02140

Abstract

This paper summarizes the current status of the DARPA Internet Activities Board (IAB) Privacy/Security Task Force's ongoing effort to enhance privacy of electronic mail transferred in the DARPA-Internet. The results of this effort will be detailed in a set of Requests for Comments (RFCs), noted here as [MP-RFC], [ALG-RFC], and [KM-RFC], and dealing, respectively, with message processing, algorithms, and key management. Official Internet RFC numbers will be assigned during the formal RFC release process.

The facilities discussed provide privacy enhancements on an end-to-end basis between originator and recipient User Agent (UA) processes, which may be implemented on heterogeneous systems. Disclosure protection, originator authenticity, and message integrity facilities are provided. A cryptographic key management approach employing RSA-based public-key certificates is defined and recommended.

Terminology

For descriptive purposes, we have used a number of standard terms defined in the OSI X.400 Message Handling System (MHS) Model per the CCITT Recommendations. The terminology has proved valuable even though the mail system considered in the current discussion is not built atop OSI protocols. This section replicates relevant definitions in order to make the terminology clear to readers who may not be familiar with the OSI MHS Model.

In the MHS model, a user is a person or a computer application. A user is referred to as either an originator (when sending a message) or a recipient (when receiving one). MH Service elements define the set of message types and the capabilities that enable an originator to transfer messages of those types to one or more recipients.

An originator prepares messages with the assistance of his or her User Agent (UA). A UA is an application process that interacts with the Message Transfer System (MTS) to submit messages. The MTS delivers to one or more recipient UAs the messages submitted to it. Functions performed solely by the UA and not standardized as part of the MH Service elements are called local UA functions.

The MTS is composed of a number of Message Transfer Agents (MTAs). Operating together, the MTAs relay messages and deliver them to the intended recipient UAs, which then make the messages available to the intended recipients. The collection of UAs and MTAs comprises the MHS.

Motivation, Approach, and Constraints

Motivation and Approach

Electronic mail is one of the most significant results of DARPA networking research, and is perhaps the result with the most wide-ranging impact on modes of human intercommunication. It is a visible and omnipresent use of networking technology, used daily as a matter of course by thousands of human users on hundreds of host computers. A wide variety of hosts implement interoperable mail service functions, supporting end users as well as relaying mail to other systems. Unfortunately, few requirements to provide privacy protection for information transferred by electronic mail have been addressed as diverse implementations have proliferated. This paper summarizes the current state of the DARPA Internet Activities Board (IAB) Privacy/Security Task Force's ongoing effort to enhance electronic mail privacy within the current Internet context.

The task force is providing a series of Requests for Comments (RFCs) to the DARPA-Internet community, presenting proposed standards for privacy-enhanced mail implementors. There are three current RFCs. RFC [MP-RFC] specifies the processing procedures to be applied to messages in order to provide privacy protection, given prior possession of appropriate cryptographic keys by originators and recipients as a necessary precondition. RFC [KM-RFC] specifies a recommended supporting key management strategy based on the use of public-key certificates and the Rivest, Shamir, Adleman (RSA) algorithm. A supporting certificate generation infrastructure is to be provided by RSA Data Security, Incorporated (RSADSI). RFC [ALG-RFC] contains definitions and references for algorithms employed in the architecture.

Services, Constraints, and Implications

Constraints and Security Services

In order to achieve applicability to the broadest possible range of Internet hosts and mail systems, and to facilitate implementation, testing, and application without the need for prior modifications throughout the Internet, two basic restrictions are imposed on the privacy enhancement mechanisms:

1. Measures must be implementable at endpoints and will be amenable to integration at the user agent (UA) level or above. Integration into the MTS (e.g., SMTP servers) will not be required. No reliance is placed on privacy-relevant service characteristics which may or may not be provided at lower protocol layers in particular hosts or networks.
2. The set of supported measures offers added value to users, enhancing rather than restricting the set of capabilities available to users. System integrity features to protect privacy enhancement software from subversion by local users cannot be assumed in general. In the absence of such features, it appears more feasible to provide facilities which enhance user services (e.g., by protecting and authenticating inter-user traffic) rather than those which enforce restrictions (e.g., inter-user access control) on user actions.

As a result of these restrictions, the following security services can be offered:

- data confidentiality
- data origin authentication
- message integrity

- if public-key key management is employed, non-repudiation of origin

but the following privacy-relevant concerns are not addressed:

- access control
- traffic flow confidentiality
- routing control
- address list accuracy
- issues relating to the casual serial reuse of PCs by multiple users
- assurance of message receipt and non-deniability of receipt
- automatic association of acknowledgments with the messages to which they refer
- message duplicate detection, replay prevention, or other stream-oriented services

Since privacy enhancement services are provided on an end-to-end basis between originators and recipients, no privacy enhancements are offered for message fields which are added or transformed by intermediate relay points. Note that the "endpoints" involved in electronic mail transfer are application layer entities within originator and recipient hosts. Although an originator and recipient may engage in a direct, real-time connection in order to transfer mail, this cannot be assumed in general. It is common fo mail to be staged and relayed at one or more sites between originator and recipient.

Two distinct privacy enhancement service options are supported:

1. an option which provides data origin authentication and message integrity
2. an option which provides data origin authentication, message integrity, and also data confidentiality through encryption

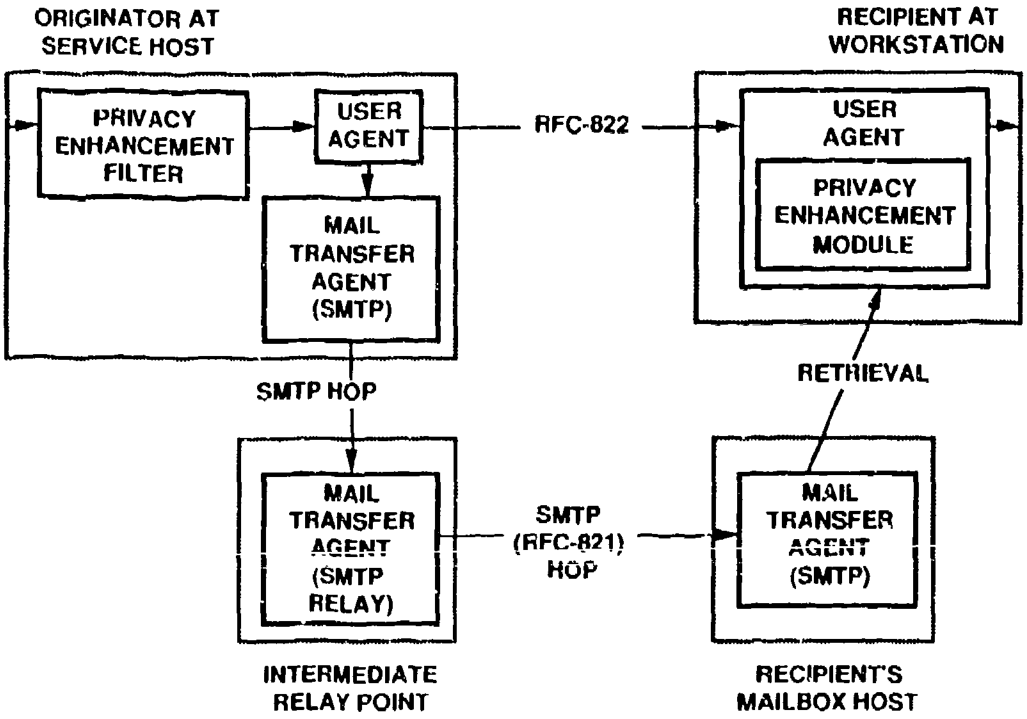
No facility for confidentiality without authentication is provided. Both options allow an originator to indicate portions of message text which are not to be enciphered; this allows non-sensitive text (as a possible example, content abstracts) to be accessed by a recipient's delegate without requiring that the delegate be privy to the recipient's personal keys.

Interoperability Issues

In keeping with the Internet's heterogeneous constituencies and usage modes, the privacy enhancement mechanisms are applicable to a broad range of Internet hosts and usage paradigms. Figure 1 illustrates an example environment. In particular, the following attributes are notable:

1. The defined mechanisms are not restricted to a particular host or operating system, but rather allow interoperability among a broad range of systems. All privacy enhancements are implemented at the application layer, independent of any privacy features which may or may not be available at lower protocol layers.
2. The defined mechanisms are compatible with Internet components which have not been enhanced to perform privacy-specific processing. Mail processing by intermediate relay hosts which do not incorporate privacy enhancement features will not be affected.

Figure 1: Environment Example


ORIGINATOR AT SERVICE HOST
PRIVACY ENHANCEMENT FILTER
USER AGENT
MAIL TRANSFER AGENT (SMTP)
RFC-822
RECIPIENT AT WORKSTATION
USER AGENT
PRIVACY ENHANCEMENT MODULE
SMTP HOP
RETRIEVAL
MAIL TRANSFER AGENT (SMTP RELAY)
SMTP (RFC-821) HOP
MAIL TRANSFER AGENT (SMTP)
INTERMEDIATE RELAY POINT
RECIPIENT'S MAILBOX HOST

3. The defined mechanisms are compatible with a range of mail transport facilities (MTAs). Within the DARPA Internet, electronic mail transport is effected by a variety of SMTP implementations. Certain sites, accessible via SMTP, forward mail into other mail processing environments (e.g., USENET, CSNET, BITNET). The privacy enhancements must be able to operate across the SMTP realm; it is desirable that they also be compatible with protection of electronic mail sent between the SMTP environment and other connected environments.

4. The defined mechanisms are compatible with a broad range of electronic mail user agents (UAs). A large variety of electronic mail user agent programs, with a corresponding broad range of user interface paradigms, is used in the Internet. In order that privacy enhancements be available to the broadest possible user community, it is desirable that the selected mechanisms be usable with the widest possible variety of existing UA programs. To facilitate deployment, it is desirable that privacy enhancement processing be incorporable into a separate program, applicable to a range of UAs, rather than requiring internal modifications to each UA with which enhanced privacy services are to be provided.

5. The defined mechanisms allow electronic mail privacy enhancement processing to be performed on PCs separate from the systems on which UA functions are implemented. Given the expanding use of PCs and the limited degree of trust which can be placed in UA implementations on many multi-user systems, this attribute can allow many users to process privacy-enhanced mail with higher assurance than a strictly UA-based approach would allow.

6. The defined mechanisms support privacy protection of electronic mail addressed to multiple recipients or to mailing lists, although protection of mail which is addressed to lists which are not expanded to individual recipients at the originator's site is limited to per-list rather than per-recipient granularity.

Message Processing Procedures

This section provides a high-level overview of the components and processing steps involved in electronic mail privacy enhancement processing.

Keying Hierarchy

A two-level keying hierarchy is used to support privacy-enhanced message transmission:

1. Data Encrypting Keys (DEKs) are symmetric keys, used for encryption of message text and for computation of message integrity check (MIC) quantities (where MIC computation algorithms requiring the use of keys are employed). DEKs are generated individually for each transmitted message; no predistribution of DEKs is needed to support privacy-enhanced message transmission.

2. Interchange Keys (IKs) are used to encrypt DEKs for transmission within messages. Ordinarily, the same IK will be used for all messages sent from a given originator to a given recipient over a period of time. Each transmitted message includes a representation of the DEK(s) used for message encryption and/or MIC computation, encrypted under an individual IK per named recipient. The representation is associated with "X-Sender-ID:" and "X-Recipient-ID:" control fields, which allow each individual recipient to identify the IK used to encrypt DEKs and/or MICs for that recipient's use. Given an appropriate IK, a recipient can decrypt the corresponding transmitted DEK representation, yielding the DEK required for message text decryption and/or MIC verification. The definition of an IK differs depending on whether symmetric or public-key cryptography is used for DEK encryption:

- When symmetric cryptography is used for DEK encryption, an IK is a single symmetric key shared between an originator and a recipient. In this case, the same IK is used to encrypt the MIC and the DEK for transmission to a recipient. Thus there is one encrypted copy of the DEK and MIC for each recipient. Version/expiration information associated with the originator and with the recipient must be concatenated in order to fully identify a symmetric IK.
- When public-key cryptography is used, the IK used to encrypt a DEK for a recipient is the public component of that recipient. Thus there is one encrypted copy of the DEK for each recipient. However, the IK used for MIC encryption is the private component of the originator, and therefore only one encrypted MIC representation is included per message, rather than one per recipient. Each of these IK components can be fully identified by an "X-Recipient-ID." or "X-Sender-ID:" field, respectively.

Encapsulation and Encoding Procedure

An encoding procedure is employed in order to represent encrypted message text in a universally transmissible form and to enable messages encrypted on one type of system to be decrypted on a different type. As Figure 2 illustrates, the header fields used for message transport are separated explicitly from those with end-to-end significance in privacy enhancement processing. As a result, transit modifications of header fields used for message transport do not disrupt privacy processing.

Four phases are involved in the encoding process:

1. (Local Form) A plaintext message is accepted in local form, using the host's native character set and line representation.
2. (Canonicalize) The local form is converted to a canonical representation, defined as equivalent to the inter-SMTP representation of message text.
3. (Encipher) The canonical representation is padded to satisfy the requirements of the encryption mode. MIC computation is performed, and if data confidentiality is selected, the padded canonical representation is encrypted.
4. (Printable Encoding) The output of the preceding step is encoded into a printable form. The printable form is composed of a restricted character set which is chosen to be universally representable across sites, and which will not be disrupted by processing within and between MTS entities.

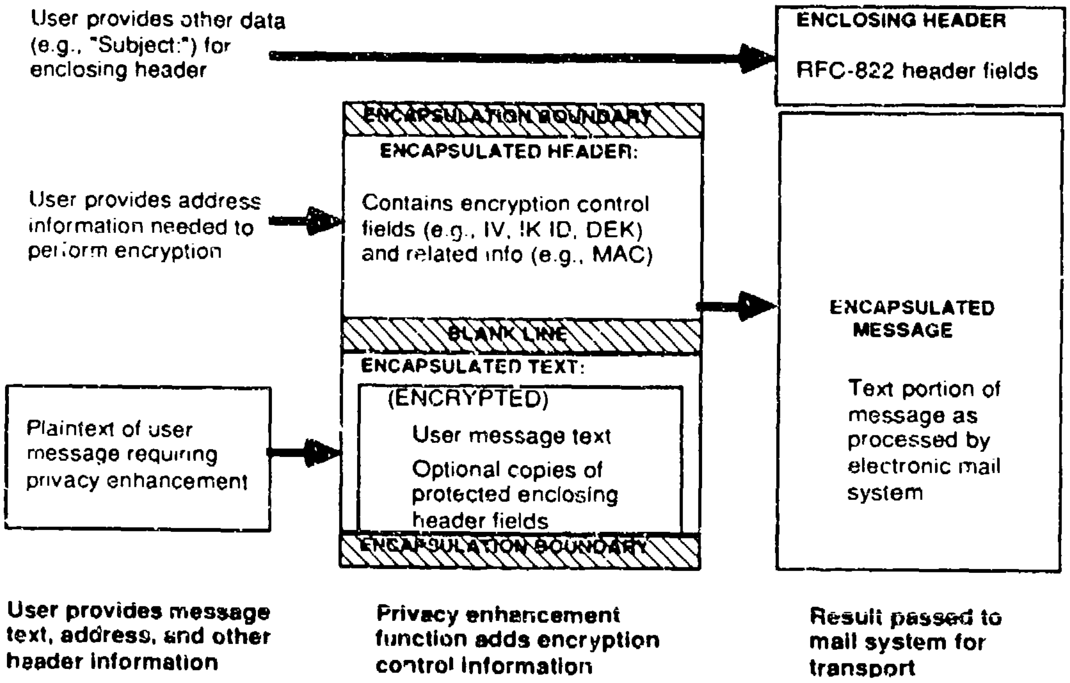
The output of the encoding procedure is combined with a set of header fields which carry cryptographic control information. The result is passed to the electronic mail system to be encapsulated as the text portion of a transmitted message. Figure 3 presents a concrete example of an encapsulated message in which public-key key management is used. Note that only one version of encrypted message text is needed in a message, independent of the number of recipients, since the message text is encrypted in a form which is usable by all recipients; only the IKs are recipient-specific, not the DEK. The set of per-recipient quantities is limited to the (relatively small) "X-Recipient-ID:" and "X-Key-Info:" encapsulated header fields.

When a privacy-enhanced message is received, the control fields within its encapsulated header provide the information which the authorized recipient requires in order to perform MIC verification and decryption on the received message text. First, the printable encoding is converted to a bitstring. If the transmitted message was encrypted, it is decrypted into the canonical representation. If the message was not encrypted, decoding from the printable form produces the canonical representation directly. The MIC is verified and the canonical representation is converted to the recipient's local form, which need not be the same as the originator's local form.

In summary, the outbound message is subjected to the following composition of transformations:

Encode(Encipher(Canonicalize(Local_Form)))

Figure 2: Message Encapsulation Mechanism


User provides other data (e.g., "Subject:") for enclosing header
ENCLOSING HEADER
RFC-822 header fields
ENCAPSULATION BOUNDARY
ENCAPSULATED HEADER:
User provides address information needed to perform encryption
Contains encryption control fields (e.g., IV, IK ID, DEK) and related info (e.g., MAC)
BLANK LINE
ENCAPSULATED TEXT:
(ENCRYPTED)
User message text
Optional copies of protected enclosing header fields
ENCAPSULATION BOUNDARY
Plaintext of user message requiring privacy enhancement
ENCAPSULATED MESSAGE
Text portion of message as processed by electronic mail system
User provides message text, address, and other header information
Privacy enhancement function adds encryption control information
Result passed to mail system for transport

The inverse transformations are performed, in reverse order, to process inbound privacy-enhanced mail.

Key Management Approach

Overview

RFC [KM-RFC] defines a recommended key management architecture based on the use of public-key certificates, supporting the message encipherment and authentication procedures defined in RFC [MP-RFC]. (Other alternative key management approaches may be defined in the future.) In the proposed architecture, a Certification Authority (CA) representing an organization applies a digital signature to a collection of data consisting of a user's public key component, various information that serves to identify the user, and the identity of the organization whose signature is affixed. This establishes a binding between these user credentials, the user's public component and the organization which vouches for this binding. The resulting signed, data item is called a certificate. The organization identified as the CA for the certificate is the "issuer" of that certificate.

Figure 3: Example Encapsulated Message

```
-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----
X-Proc-Type: 3,ENCRYPTED
X-DEK-Info: DES-CBC,F8143EDE5960C597
X-Sender-ID: Feldman@ccy.bbn.com::
X-Certificate:
 jHU1BLpvXR0UrUzYbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIk
 YbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU1BLpvXR0UrUz
 agV2IzUpk8tEjmFjHU1BLpvXR0UrUz/zxB+bATMtPjCUWbz8Lr9wloXIkYbkNpk0
X-Issuer-Certificate:
 TMtPjCUWbz8Lr9wloXIkYbkNpk0agV2IzUpk8tEjmFjHU1BLpvXR0UrUz/zxB+bA
 IkjHU1BLpvXR0UrUzYbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloX
 vXR0UrUzYbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU1BLp
X-MIC-Info: RSA-MD2,RSA,
 5rDqUcM1K1Z6720dcBWGGsDLpTpSCnpotJ6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz
X-Recipient-ID: Feldman@ccy.bbn.com:RSADSI:3
X-Key-Info: RSA,
 1BLpvXR0UrUzYbkNpk0agV2IzUpk8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU
X-Recipient-ID: privacy-tf@venera.isi.edu:RSADSI:4
X-Key-Info: RSA,
 NcUk2jHEUSoH1nvNSIWL9MLLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72oh

LLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M
8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHU1BLpvXR0UrUzYbkNpk0agV2IzUpk
J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcM1K1Z6720dcBWGGsDLpTpSCnpot
dXd/H5LMDWnonNvPCwQUHt==
-----PRIVACY-ENHANCED MESSAGE BOUNDARY-----
```

The contents of the certificate are as follows:

- Serial Number
- Issuer Name
- Subject Name
- Validity Period Information
- Subject Public Component and Associated Algorithm Identifier
- Certificate Signature (encrypted hash) and Associated Algorithm Identifier

In signing the certificate, the CA vouches for the user's identification, especially as it relates to the user's affiliation with the organization. The digital signature is affixed on behalf of that organization and is in a form which can be recognized by all members of the privacy-enhanced electronic mail community. Once generated, certificates can be stored in directory servers, transmitted via unsecure message exchanges, or distributed via any other means that make certificates easily accessible to message originators, without regard for the security of the transmission medium.

Prior to sending an encrypted message, an originator must have acquired a certificate for each recipient and must have validated these certificates. Briefly, validation is performed by checking the digital signature in the certificate, using the public component of the issuer whose private component was used to sign the certificate. The issuer's public component is made available via some (integrity assured) out of band means or is itself distributed in a certificate to which this validation procedure is applied recursively.

Once a certificate for a recipient is validated, the public component contained in the certificate is extracted and used to encrypt the data encryption key (DEK) that is used to encrypt the message itself. The resulting encrypted DEK is incorporated into the "X-Key-Info:" field of the message header. Upon receipt of an encrypted message, a recipient employs his private component to decrypt this field, extracting the DEK, and then uses this DEK to decrypt the message.

In order to provide message integrity and data origin authentication when public-key key management is used, the originator generates a MIC, signs (encrypts) the MIC using the private component of his public-key pair, and includes the resulting value in the message header in the "X-MIC-Info:" field. The certificate of the originator is also included in the header in the "X-Certificate:" field. Upon receipt of a privacy enhanced message, a recipient validates the originator's certificate, extracts the public component from the certificate, and uses that value to recover (decrypt) the MIC. The recovered MIC is compared against the locally calculated MIC to verify the integrity and data origin authenticity of the message.

Scope and Restrictions

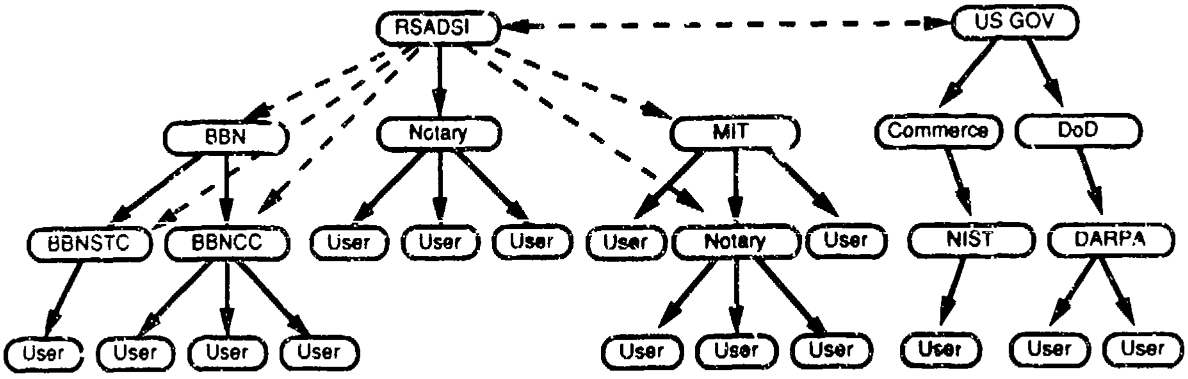
While X.509 defines the concept of certification path, allowing recursive validation of a chain of certificates, our proposed architecture imposes additional conventions for certification paths beyond those required by X.509 or by its underlying cryptographic technology. The decision to impose these conventions is based in part on constraints imposed by the status of the RSA cryptosystem within the U.S. as a patented algorithm, and in part on the need for an organization to assume operational responsibility for certificate management in the current (minimal) directory system infrastructure for electronic mail.

Thus, for example, we propose a system in which the user certificates represent the leaves in a shallow certification hierarchy (tree). Figure 4 illustrates an example certification hierarchy consistent with the architecture. In this example each oval represents a certificate for a specified organization or (generic) user. Note that only organizations act as issuers in this architecture; a user certificate may not appear in a certification path except as a terminal node in the path. Each line in the figure points to a certificate which is issued by the organization from which the line emanates. The solid lines mirror paths in the (X.500) naming hierarchy, whereas dashed lines indicate additional certification relations not implied by the naming hierarchy. (Thus organizations pointed to by dashed lines are generally identified as subjects in more than one certificate.) The line between RSADSI and the U.S. Government demonstrates "cross-certification," i.e., each organization has issued a certificate vouching for the other. This facilitates interoperation across jurisdictional boundaries, as discussed later. The conventions noted above, though not required by X 500, contribute to simplified validation of user certificates.

RFC [KM-RFC] proposes that RSADSI act as the generator of certificates on behalf of most organizations, with two notable exceptions. First, the U.S. Government has royalty-free use of the RSA algorithm and thus may establish a certification facility on behalf of its organizations, personnel, etc.. Second, organizations outside of the U.S. are not bound by the

RSA patent and thus certification facilities will probably be established in other countries. The role RSADSI will play for most U.S. organizations can be effected in a "transparent" fashion so that the organizations appear to be the issuers with regard to certificate formats and validation procedures, while imposing accounting controls in support of licensing. This avoids the need for an organization to establish the stringent accounting mechanisms and enter into more elaborate legal agreements that are required if an organization assumes responsibility for certificate generation in support of its user community. It also establishes a uniform level of trust in the certificate generation procedure that would be difficult to obtain in a more distributed environment.

Figure 4: Certification Paths



RFC [KM-RFC] specifies procedures by which users order certificates either directly from RSADSI or via a representative in an organization with which the user holds some affiliation (e.g., the user's employer or educational institution). Syntactic provisions are made which allow a recipient to determine, to some granularity, which identifying information contained in the certificate is vouched for by the certificate issuer. In particular, organizations will usually be vouching for the affiliation of a user with that organization and perhaps a user's role within the organization, in addition to the user's name. In other circumstances, a certificate may indicate that an issuer vouches only for the user's name, implying that any other identifying information contained in the certificate may not have been validated by the issuer. These semantics are beyond the scope of X.509, but are not incompatible with that recommendation.

The certificate issued to a user for a $25 biennial fee grants to the user identified by that certificate a license from RSADSI to employ the RSA algorithm for certificate validation and for encryption and decryption of DEKs, MICs and message digests in this electronic mail context. No use of the algorithm outside the scope defined in this RFC is authorized by this license. The license granted by this fee does not authorize the sale of software or hardware incorporating the RSA algorithm; it is an end-user license, not a developer's license.

Certificate Ordering Procedures

A user may order a certificate in two ways: through the user's affiliation with an organization or directly through RSADSI. In either case, a user will be required to send a paper order to RSADSI on a form containing the following information:

1. Distinguished Name elements (e.g., full legal name, organization name, etc.)
2. Postal address
3. Internet electronic mail address
4. A one-way hash function, binding the above information to the user's public component

If the user is not affiliated with an organization which has established its own "electronic notary" capability, an organization notary (ON) as discussed in the next subsection, then this paper form also must be notarized by a Notary Public. If the user is affiliated with an organization which has established one or more ONs, the paper form need not carry the endorsement of a Notary Public. Concurrent with the paper application, the user must send the information outlined above, plus his public component, either to his ON, or directly to RSADSI if no appropriate ON is available to the user. Transmission between a user and an ON is a local matter, but we expect electronic mail will also be the preferred option in many circumstances.

Organizational Notaries

An organizational notary is an individual who acts as a clearinghouse for certificate orders originating within an administrative domain such as a corporation or a university. An ON represents an organization or organizational unit (in X.500 naming terms), and is assumed to have some independence from the users on whose behalf certificates are ordered. An ON will be constrained (by mechanisms implemented by RSADSI) to ordering certificates properly associated with his domain. For example, an ON for BBN would not be able to order certificates for users affiliated with MITRE nor vice versa. Similarly, if a corporation such as BBN were to establish ONs on a per-subsidiary basis (corresponding to organization units in X.500 name parlance), then an ON for BBN Communications Corp. would not be allowed to order a certificate for a user who claims affiliation with BBN Systems and Technologies Corp. (see Figure 4).

It can be assumed that the set of ONs changes relatively slowly and that the number of ONs is relatively small in comparison with the number of users, so a more costly and better-assured process may reasonably be associated with ON accreditation than with per-user certificate ordering. Restrictions on the range of information which an ON is authorized to certify are established as part of this more elaborate registration process.

An ON is responsible for establishing the correctness and integrity of information incorporated in an order, and will generally vouch for (certify) the accuracy of identity information at a granularity finer than that provided by a Notary Public. Although it is not feasible to enforce uniform standards for the user certification process across all ONs, we anticipate that organizations will endeavor to maintain high standards in this process in recognition of the "visibility" associated with the identification data contained in certificates. An ON also may constrain the validity period of an ordered certificate, restricting it to less than the default two year interval imposed by RSADSI.

An ON participates in the certificate ordering process by accepting and validating identification information from a user and forwarding it to RSADSI. The ON accepts the ordering information described earlier, plus the user's public component, from a user. (Each user locally generates his own public and private component pair. He holds the private component secret, so that neither his ON, RSADSI, nor any other user is ever privy to this value.) The ON sends a privacy-enhanced electronic message to RSADSI, vouching for the correctness of the binding between the public component and the identification data. Thus, to support this function, each ON will hold a certificate as an individual user within the organization which he represents. RSADSI will maintain a database which identifies the users who also act as ONs and which will specify constraints on credentials which each ON is authorized to certify.

Certification Authorities

In X.509, a CA is defined as "an authority trusted by one or more users to create and assign certificates". In X.509, however, there is no requirement that a CA be a distinguished entity or that a CA serve a large number of users, as envisioned in the proposed privacy-enhanced mail architecture. Rather, any user who holds a certificate can, in the X.509 context, act as a CA for any other user. We have chosen to restrict the role of CA in this electronic mail environment to organizational entities, to simplify the certificate validation process, to impose semantics which support organizational affiliation as a basis for certification, and to facilitate license accountability.

In the proposed architecture, individuals who are affiliated with (registered) organizations will go through the process described previously, in which they forward their certificate information to their ON for certification. The ON will, based on local procedures, verify the accuracy of the user's credentials and forward this information to RSADSI using privacy-enhanced mail to preserve the integrity and authenticity of the information. RSADSI will carry out the actual certificate generation process on behalf of the organization represented by the ON. It is the identity of the organization which the ON represents, not the ON's identity, which appears in the issuer field of the user certificate. Therefore it is the private component of the organization, not the ON, which is used to sign the user certificate.

In order to carry out this procedure RSADSI will serve as the repository for the private components associated with certificates representing organizations or organizational units (but not individuals). In effect the role of CA will be shared between the organizational notaries and RSADSI. This shared role will not be visible in the syntax of the certificates issued under this arrangement nor is it apparent from the validation procedure one applies to these certificates. In this sense, the role of RSADSI as the actual generator of certificates on behalf of organizations is transparent to this aspect of system operation. RSADSI merely appears as an organization which happens to have "cross-certified" most other organizations in the U.S (non-government) naming hierarchy. Similarly, any U.S. Government CAs and foreign CAs will cross-certify RSADSI, and vice versa, to permit uniform certificate validation procedures across the administrative boundaries implied by these CAs.

RSADSI has offered to operate a service in which it serves as a CA for users who are not affiliated with any organization or who are affiliated with an organization which has not opted to establish an organizational notary. To distinguish certificates issued to such "non-affiliated" users the distinguished string "Notary" will appear as the OrganizationalUnitName of the issuer of the certificate. Thus not only RSADSI but any other organization which elects to provide this type of service to non-affiliated users may do so in a standard fashion. Thus a corporation might issue a certificate with the "Notary" designation to students hired for the summer, to differentiate them from full-time employees. At least in the case of RSADSI the standards for verifying user credentials that carry this designation will be well known and widely recognized (e.g., Notary Public endorsement). Figure 4 illustrates how the "Notary" convention could be employed by both RSADSI and MIT.

Revoked Certificate Lists

X.509 states that it is a CA's responsibility to maintain:

1. a time-stamped list of the certificates it issued which have been revoked
2. a time-stamped list of revoked certificates representing other CAs

There are two primary reasons for a CA to revoke a certificate, i.e., suspected compromise of a secret component (invalidating the corresponding public component) or change of user affiliation (invalidating the Distinguished Name). As described in X.509, "hot listing" is one means of propagating information relative to certificate revocation, though it is not a perfect mechanism. In particular, an X.509 Revoked Certificate List (RCL) indicates only the age of the information contained in it; it does not provide any basis for determining if the list is the most current hot list available from a given CA.

To help address this concern, the proposed architecture establishes a format for a RCL in which not only the date of issue, but also the next scheduled date of issue is specified. (This is a deviation from the format specified in X.509.) When that date arrives a new RCL must be issued, even if there are no changes in the list of entries. Thus each CA can independently establish and advertise the frequency with which hot lists are issued by that CA. This does not preclude issuance on a more frequent basis, in case of some emergency, but no mechanisms are provided for alerting users that such an unscheduled issuance has taken place. This scheduled RCL issuance convention allows users or UAs to determine whether a given RCL is "current."

The X.509 recommendation previously required revoked certificate lists to contain entire certificates. The recommendation now calls for each hot list to contain the serial numbers assigned to the revoked certificates. The inclusion of a serial number in each certificate, unique for all certificates issued by the indicated CA, and the corresponding change to the revoked certificate list format, were a direct result of suggestions offered by members of the Task Force. It is gratifying to see these suggestions incorporated into the CCITT/ISO standards process within the course of one year.

Status

As of this writing, the message processing procedures RFC [MP-RFC] is about to be released in its third version and the first versions of the companion key management RFC [KM-RFC] and algorithms RFC [ALG-RFC] are also slated to be released shortly. Successful interoperability tests, among several sites, have been performed to validate the privacy-enhanced message processing specifications. Tests employing the certificate-based key management technology have been carried out at one site. We anticipate distribution of a "reference implementation," integrated into the MH mail system for use with Berkeley UNIX ™ and

™ UNIX is a trademark of AT&T.

derived operating systems, throughout the Internet in the Fall of 1989. The supporting key management infrastructure described herein also should be in place by late 1989.

Acknowledgments

This paper is a result of a series of IAB Privacy/Security Task Force meetings and of RFCs generated as a result of those meetings. Particular thanks are due to the following task force members and meeting guests for their comments and contributions: David Balenson, Curt Barker, Jim Bidzos, Matt Bishop, Morrie Gasser, Russ Housley, Dan Nessett, Mike Padlipsky, Rob Shirey, Miles Smid, and Steve Wilbur.

The authors' participation in IAB Privacy/Security Task Force activities (for the case of John Linn, during prior employment at BBN Communications Corporation) has been supported by the Defense Advanced Research Projects Agency through tasking under Contract No. F29601-87-C-0086, Network-Oriented Systems. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

References

CCITT, Recommendation X.400, "Message Handling: System and Service Overview".

CCITT, Recommendation X.402, "Message Handling Systems: Overall Architecture".

CCITT, Recommendation X.411, "Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures".

CCITT, Recommendation X.500, "The Directory - Overview of Concepts, Models, and Services".

CCITT, Recommendation X.509, "The Directory - Authentication Framework".

Crocker, D. H., Network Working Group Request for Comments (RFC) 822, August 13, 1982, "Standard for the Format of ARPA Internet Text Messages".

Postel, J., Network Working Group Request for Comments (RFC) 821, August 1982, "Simple Mail Transfer Protocol".

Glossary

CA: Certification Authority

DARPA: Defense Advanced Research Projects Agency

DEK: Data Encrypting Key

IAB: Internet Activities Board (of DARPA)

IK: Interchange Key

MHS: Message Handling System

MIC: Message Integrity Check

MTA: Message Transfer Agent

MTS: Message Transfer System

O/R: Originator/Recipient

ON: Organizational Notary

RFC: Request for Comments

RSA: Rivest, Shamir, Adleman (public-key encryption algorithm)

RSADSI: RSA Data Security, Incorporated

SMTP: Simple Mail Transfer Protocol

UA: User Agent

KEY MANAGEMENT AND ACCESS CONTROL FOR AN ELECTRONIC MAIL SYSTEM

Martha Branstad
W. Curtis Barker
Pamela Cochrane
David Balenson

Trusted Information Systems, Inc.
Glenwood, MD 21738

1.0 Introduction

Traditional end-to-end encryption systems are implemented in independent hardware components accessed at or below the interface between the network and transport functions of the host computer. This approach assures a relatively inviolate domain for high integrity cryptography. However, placement of cryptographic services below the transport layer constrains the ability to provide user-to-user cryptographic protection needed to support a secure electronic mail system. A secure electronic mail system with encryption below the transport layer requires substantial modification to existing networks protocols and the inclusion of large amounts of frequently changed user information in outboard cryptographic modules but still retains a critical dependency upon the host system software to establish user identity and the security level of mail messages. A reasonable alternative for secure electronic mail is to place encryption at a higher level in the host system. This placement implies software control of cryptographic functions. In such a system the recognition that cryptography is required for protection of a message, the isolation of header from message text, and the correspondence of user with key identifier are all performed by the system software. Systems with cryptographic control software demand the assurance and support of trusted system technology, as well as that of cryptographic technology, making them classic examples of Information Security (INFOSEC) products.

Trusted Information Systems, Inc. (TIS), under Defense Advanced Research Projects Agency (DARPA) funding is investigating means for providing Information Security to the Internet. User-to-user protection of electronic mail and transport layer protection are being studied. The research is integrating techniques for protection of sensitive information within a computer with those for information in transit between computers, attempting to meet both communications security and trusted systems objectives. An initial proof-of-concept prototype, the Embedded Network Security (ENS) system, is being developed in which cryptography and key management are embedded in the software of a trusted system, TMach [1]. The ENS system provides confidentiality, message integrity, and source authentication services in conjunction with electronic mail and transport services. This paper examines key management and access control services associated with the ENS Trusted Mail (TMail) system, indicating how both encryption and trusted system functionality provide protection. The interaction between trusted system protection mechanisms and those supplied by cryptographic techniques is highlighted, illustrating how INFOSEC products make strong demands on both disciplines.

References

[1] Barker, W. C., et al., "Embedding Cryptography into a Trusted Mach System," Proceedings of the Fourth Aerospace Computer Security Conference, December 1988.

[2] Nelson, R., "SDNS Services and Architecture," Proceedings of the 10th National Computer Security Conference, September 1987.

[3] Linn, J., Request for Comments (RFC) 1040, Privacy Enhancement for Internet Electronic Mail: Part I: Message Encipherment and Authentication Procedures, January 1988.

[5] American National Standard Data Encryption Algorithm, ANSI X3.92-1981, American National Standards Institute, Approved 30 December 1980.

[6] Branstad, M., et al., "Trusted Mach Design Issues," Proceedings of the Third Aerospace Computer Security Conference, December 1987.

[7] Branstad, M., et al., "Access Mediation in a Message Passing Kernel, Proceedings of the 1989 IEEE Symposium on Security and Privacy, May 1989.

[8] Rashid, R., "Threads of a New System," Unix Review, Vol.4, No. 8, August 1986.

[9] Federal Information Processing Standard Publication 113, Computer Data Authentication, May 1985.

[10] Department of Defense, Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December 1985.

[11] DoD Computer Security Center, Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-003-85, June 1985.

June 30, 1989

A TOKEN BASED ACCESS CONTROL SYSTEM FOR COMPUTER NETWORKS

Miles Smid, James Dray, and Robert B.J. Warnar

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

Supported by the Defense Advanced Research Projects Agency under order 6373.

ABSTRACT

This paper describes a Token Based Access Control System (TBACS) developed by the Security Technology Group of the National Institute of Standards and Technology (NIST). TBACS replaces traditional password based access control systems which have often failed to prevent logins by unauthorized parties. A user's access to network computers and resources is mediated by a smart token implementing a transparent cryptographic three-way handshake with the target computer. The token's onboard processor and memory are exploited to provide sophisticated security mechanisms in a portable device. In addition to access control, the TBACS token may be used for random number generation, cryptographic key generation, data encryption, data authentication, and secure data storage.

I. INTRODUCTION

A computer is a valuable resource which should be protected. The information within the computer should be protected from unauthorized disclosure and modification, and the computing power should be limited to authorized users. The recent rash of computer viruses and the previous successes of hackers in gaining access to computer systems indicates that many computers are not properly protected. Inadequate protective measures are not justified by the statement that the computers contained only unclassified data. The failure to control access to a computer's resources (whether classified or not) has serious consequences.

Most computer systems attempt to protect their resources by authenticating the identity of each user attempting to login. Once the user's identity is established, the system then controls the access of the user to resources based upon some predetermined access control policy.

Unfortunately, as we progressed from localized stand-alone systems to distributed processing systems on large networks, it became easier to subvert the traditional protective mechanisms. At one time, access to a computer's resources could be controlled by limiting the access to the room where the computer was physically located. Today computers are networked so that remote users may take advantage of distributed resources without having to be physically co-located. We now rely on password systems which are not up to the task of protecting computer resources.

In theory, there are three types of information for authenticating the identity of computer users [1,2]:

1. Something the user KNOWS (such as a password)

2. Something the user POSSESSES (such as a token), and

3. Some PHYSICAL CHARACTERISTIC of the user (such as fingerprints or other biometric data).

In practice, most computer systems use only the first type of information (e.g. passwords) to authenticate user identities. Password systems predominate because they are inexpensive and they appear, upon first examination, to be easy to use. Password systems do not provide the highest level of security. If properly implemented, password systems can provide effective security [3]. However, these systems are seldom properly implemented. Time and time again we hear about cases where a user selected a trivial password, the user wrote down or shared a password, the operating system debuggers left well known passwords in the system, or the passwords were transmitted over an unprotected channel in the clear.

The owners and users of most computer systems have not been willing to suffer the expense and the effort associated with token and biometric based authentication systems. A major exception to this rule has been the retail banking community. Most users of Automatic Teller Machines are accustomed to the fact that in order to obtain their money, they must produce a bankcard as well as a password known as a Personnel Identification Number (PIN). These systems have had some security problems but it is generally acknowledged that they are superior to password-only applications. If all computer systems required tokens for access, most hackers would be prevented from entering systems to which they were not authorized.

The cost of electronic technology has decreased substantially over the last ten years making biometric based authentication much more feasible. Biometric systems are now being considered for limited high security applications. Although, biometric systems still have a significant cost, they may some day become the standard in authentication systems.

Password systems alone are not as easy to use, in a secure manner, as some previously thought.

1. If passwords are randomly generated, they are written down. If passwords are generated by humans, they can often be guessed.

2. If a user needs a different password for each computer to which access is permitted, then the user becomes frustrated and writes the passwords down.

3. If the communications link between the user terminal and the host computer is unprotected, then a line tapper can determine the password and later login as the user.

This paper describes a Token Based Access Control System (TBACS) which is being developed by the National Institute Of Standards and Technology (NIST). The first version of TBACS will use a single user password and a smart token containing cryptography to reduce or eliminate several of the drawbacks associated with password systems. Later versions may employ biometrics for increased security as that technology becomes more cost effective.

II. DESIGN REQUIREMENTS

TBACS was designed by NIST to satisfy the following requirements:

1. TBACS shall be easy to use. A TBACS user only needs to remember one password for all computer systems to which the user has access. The TBACS user authenticates to the token via the password, but does not have to type any challenges or responses. The token authenticates the user to all computers (the user workstation and remote hosts).

2. TBACS shall implement the mechanisms for cryptographic authentication as well as cryptographic key storage on the token itself. The closer the security to the user the better. Once inserted, keys will not leave the token.

3. TBACS shall be consistent with existing government and American National Standards Institute (ANSI) standards. The token implements the Data Encryption Standard (DES) cryptographic algorithm specified in Federal Information Processing Standard (FIPS) 46 [4], and could also be used to authenticate computer data and messages as specified in FIPS 113 [5], ANSI X.9 [6], and ANSI X9.19 [7]. TBACS is consistent with Draft American National Standard for Financial Institution Sign-On Authentication for Wholesale Financial Systems (ANSI X9.26) [8].

4. TBACS tokens shall have the capability to store additional information such as sensitivity labels and other access control information.

5. TBACS shall be capable of serving multiple security needs. Although TBACS token is primarily designed for user authentication, it can also be used for random number generation, cryptographic key generation, low speed encryption, low speed Message Authentication Code calculation, and secure data storage. Future versions of TBACS could function with biometric authentication devices.

NIST decided that the best way to ensure that all its requirements were met was to specify the exact command set that the token would implement. In addition to implementing the desired capabilities, security could be improved because only a limited well defined command set was allowed.

III. SYSTEM DESCRIPTION

The NIST secure computer network research model consists of a Sun workstation connected to an Ethernet with one or more hosts (Figure 1). Each computer on the net is interfaced to a token reader/writer system. Access to the net is granted after a predefined sequence of authentications have been completed between the user, the token, the workstation, and any selected computers on the network.

When the token is inserted into the reader/writer, a C-language program in the workstation starts the login sequence by making calls to commands implemented in the token. The user is prompted for the user identifier (ID) and a Personal Identification Number (PIN) which, if correct, authenticates the user to the token. From this point on, the token acts for the user to perform a mutual DES based cryptographic authentication with the workstation and any other hosts to which the user is permitted access (Figure 2).

A. Hardware

The smart token consists of a plastic carrier containing a microprocessor and nonvolatile memory. The carrier has the same major dimensions as a standard credit card, with six recessed metallic contacts along one edge. The reader/writer provides the following electrical connections to the token via the six contacts: power, ground, hardware reset, clock, serial data in, and serial data out. The reader/writer connects to the workstation through a standard asynchronous serial communications port, eliminating the need for a custom communications interface.

TBACS is designed to operate with workstations operating under UNIX (TM), which implement the DES in hardware using a crytographic chip set. The use of personal computers (PCs) as workstations will also be supported.

B. Software

NIST designed a set of sixteen individual token commands. Several of these commands must be executed in a predefined sequence. The sequence is controlled by a set of flags which are checked each time a command is performed. If the flags are not in the expected state, the system will return an error and the current command will not be executed.

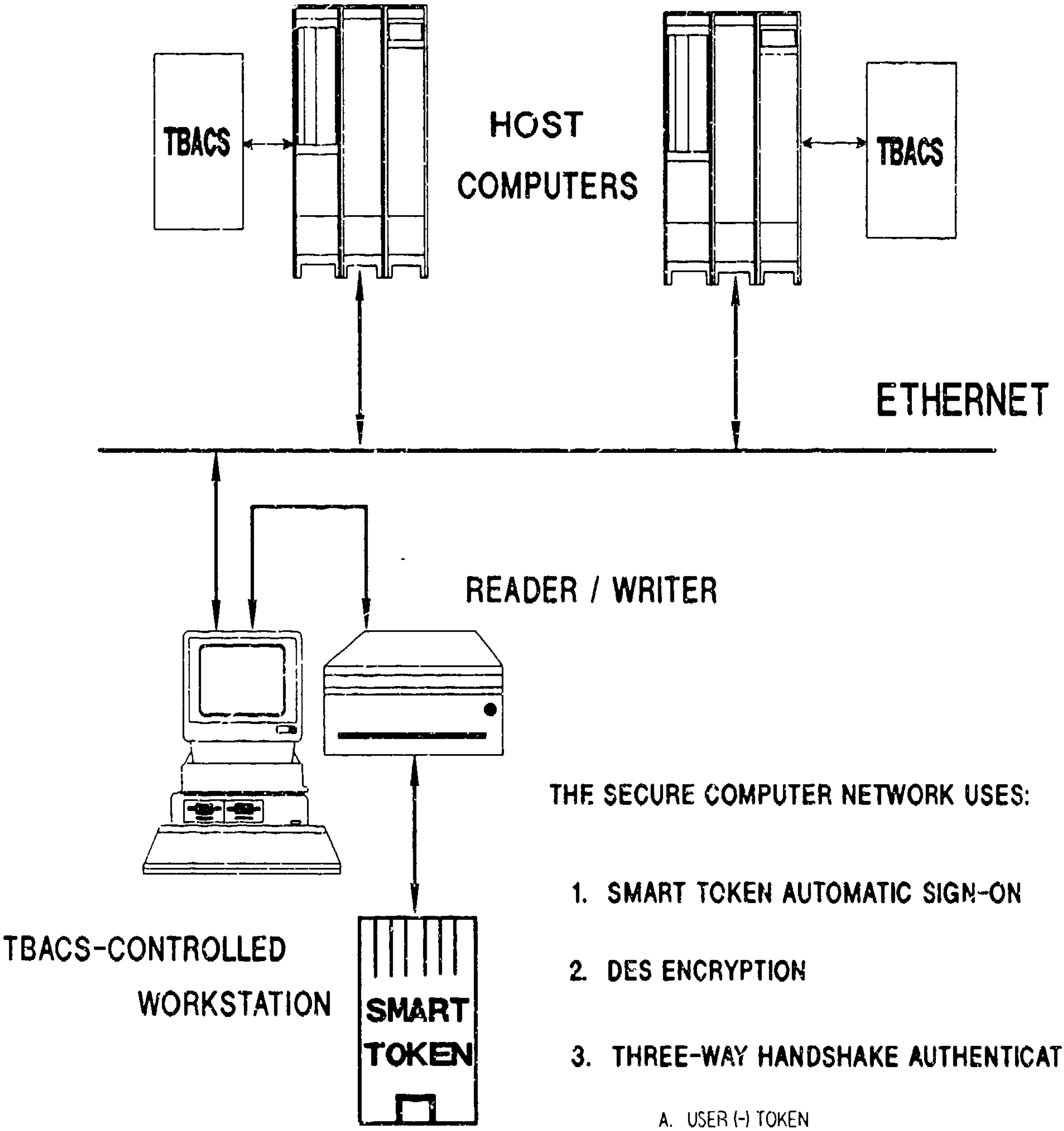
The commands are grouped into three classes: the Security Officer (SO) commands, the user to workstation commands, and the user to host commands. The SO commands provide for the initialization of tokens including the loading of cryptographic keys, host IDs, and PINs. The token is ready to be issued to the user after the SO has completed the "loading process".

The token key table contains the host IDs and the

Figure 1.

NIST Secure Computer Network Model

NIST SECURE COMPUTER NETWORK RESEARCH



TBACS

HOST COMPUTERS

TBACS

ETHERNET

READER / WRITER

TBACS-CONTROLLED WORKSTATION

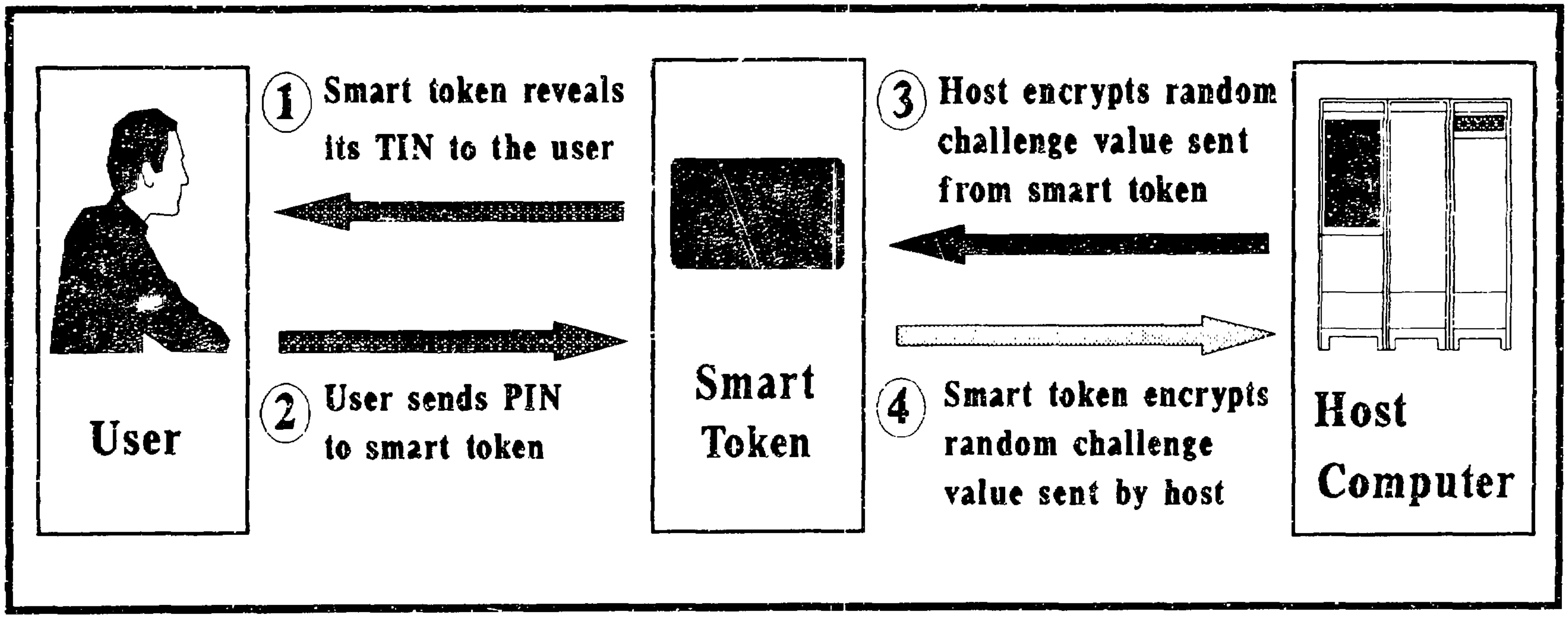
SMART TOKEN

THF SECURE COMPUTER NETWORK USES:

1. SMART TCKEN AUTOMATIC SIGN-ON
2. DES ENCRYPTION
3. THREE-WAY HANDSHAKE AUTHENTICATION
   - A. USER (-) TOKEN
   - B. WORKSTATION (-) TOKEN
   - C HOST (-) TOKEN

Figure 2.

Mutual Authentication in the TBACS


User
1 Smart token reveals its TIN to the user
2 User sends PIN to smart token
Smart Token
3 Host encrypts random challenge value sent from smart token
4 Smart token encrypts random challenge value sent by host
Host Computer

corresponding cryptographic keys. The design supports 100 cryptographic keys for 100 different hosts connected on the Ethernet. The host IDs and keys are part of a set of the parameters that must be entered by the SO during token initialization. The token uses the keys in this table to perform encryption and decryption processing during workstation and host authentications.

A software simulation program has been written in C which implements the operations of the token as defined by its command set. The simulation forms the main part of the detailed system specification and is used to specify the system. The simulation consists of sixteen functions, one for each token command, plus a small number of internal functions. The total simulation consists of about 2500 lines of code.

The workstation software must interact with the user token through the reader/writer. It must also act as an intermediary in the authentications between the user and the token and between the token and the workstation cryptographic module. If the user wishes to login to a remote host, the workstation software must implement the necessary communications protocols and prompt the token to perform authentication functions as required. The workstation will have security officer controlled software for enrolling new users. The workstation software will store or be able to calculate keys for all valid workstation users.

The software of the network host computers must be able to communicate with the user workstation. Like the workstation, it must have security officer controlled software for enrolling new users and maintaining keys.

IV. AUTHENTICATION PROCESSES

In order for a user to gain access to computing resources on a network using TBACS, a series of authentications between the smart token, the user, and various host computers must be performed. TBACS selectively controls access to all computers on the network, including the user's local workstation. By taking advantage of the processing capabilities of the smart token, the login process can proceed transparently to the user while providing a high level of security. The DES algorithm, operating firmware, and critical data are stored internally on the smart token.

A. USER/TOKEN AUTHENTICATIONS

When a user begins the login process on a workstation, the user should have some means of determining the identity of the token. A program called the "login manager" is executed on the workstation when the user initiates a login, and is responsible for mediating the required series of authentications between the user, the token, and the workstation. The first step performed by the login manager is to request the token identification number (TIN) from the token and display it on the user's screen for visual verification. The user can choose to either continue the login process or abort. If the user chooses to continue, the user must prove his identity to the token. The login manager prompts the user for the user PIN, which is then encrypted by the workstation and transmitted to the token along with the user ID. The token decrypts the user PIN and uses it as the key to encrypt the user identity. The result is then compared to the value stored on the token, and if these values match the token accepts the identity of the user as authentic. From this point on, TBACS uses the token to authenticate the user's identity to other computers.

B. THREE-WAY HANDSHAKE

The three-way handshake is the authentication protocol used between the token and the workstation and between the token and the remote host(s). This protocol allows each party to prove that it possesses the same cryptographic key as the other party [9] (Figure 3). This protocol works as follows:

1. Party A generates a 64-bit random number and transmits it to party B.

2. Party B encrypts the random number using its DES key, generates a second random number, and transmits it to party A.

3. Party A decrypts the first number and verifies the result. Party A then encrypts the second random number and transmits it to party B.

4. Party B decrypts and verifies the second random number. At this point, each party is satisfied that the other party possesses the DES key corresponding to the claimed identity. Therefore both parties are implicitly authenticated.

C. USER/WORKSTATION AUTHENTICATIONS

After the user and token authenticate to each other, the token must authenticate to the workstation. To perform the authentications between the workstation and the token, the login manager requests a random number from the token. The three-way handshake then proceeds with the token acting as party A and the workstation as party B. If this handshake is completed successfully, the login manager terminates and the user is logged in to the system.

D. USER/REMOTE HOST AUTHENTICATIONS

At some point during a session, the user may decide to connect to a remote host via the network. The user activates a remote login manager, which requests a table of the allowed TBACS hosts for this user from the token and displays this table in a menu format. After the user selects the desired remote host from this menu, the remote login manager connects to the remote login server on the remote host. At this point, the local remote login manager acts primarily as a transparent communications path between the token and the remote login server. The token is provided with the host ID, which it uses to select the proper key for subsequent cryptographic operations. The steps of the three-way handshake are then performed between the token and the remote login server on the remote host. Finally, the remote login server terminates and the standard remote login process connects the user to the remote host.

E. SEQUENCE CONTROL

In order for the steps which accomplish the authentications required by TBACS to function, some mechanism for ensuring that these steps are executed in the correct order must be provided. This is a critical design consideration, since the overall security of the system is dependent on this order. TBACS controls the order in which the authentication steps are executed through a set of "sequence flags" stored internally on the token. These flags are individual bits in the token's memory, which are set in sequence upon successful completion of each step. The flags are checked at the beginning of the next step. Since the flags and the mechanism for controlling them are internal to the token and no external access is provided, it is difficult to defeat the correct sequencing of steps.

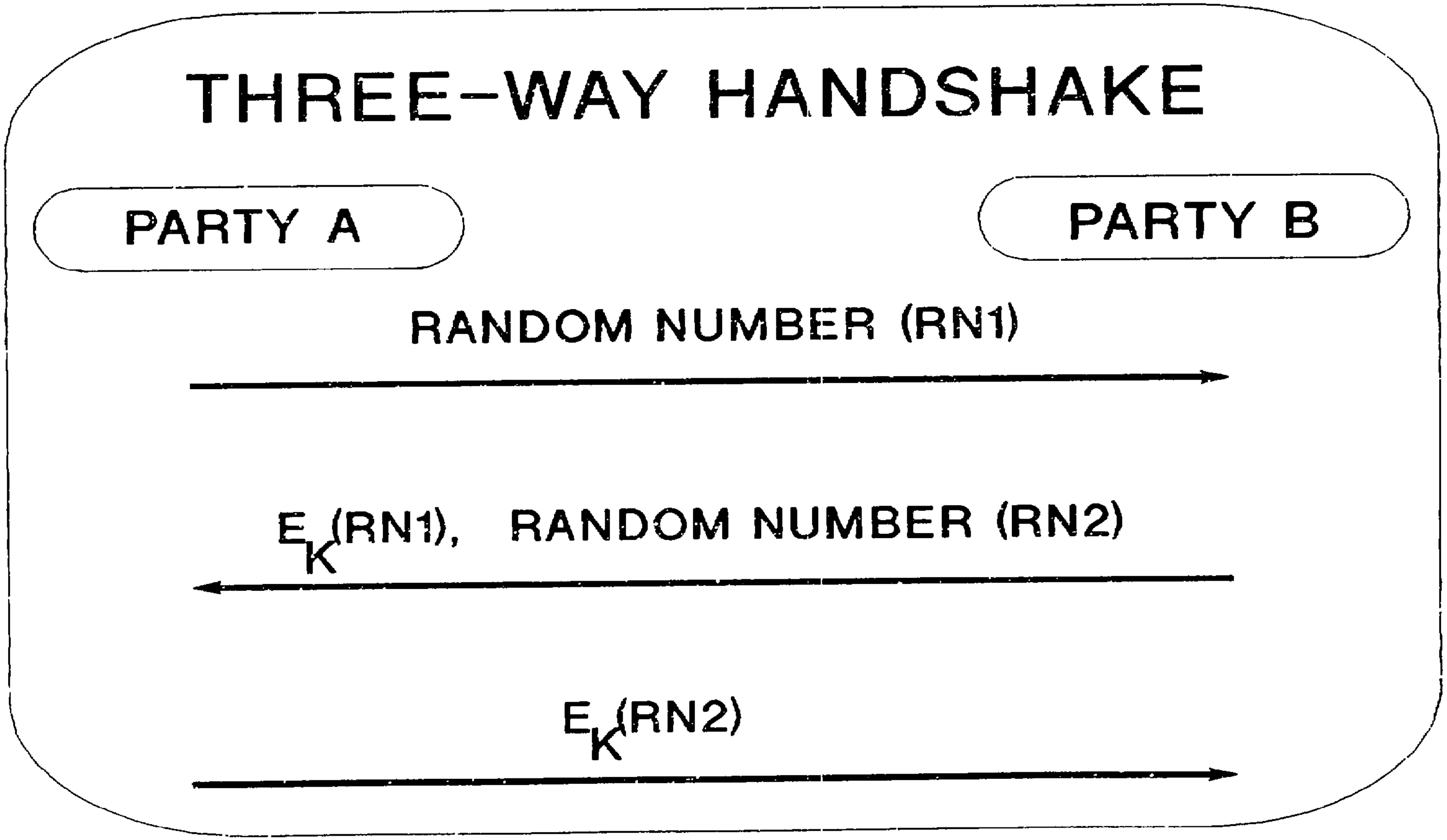
F. TOKEN DEACTIVATION

In addition to sequence control, the TBACS token is capable of deactivating itself when certain conditions are detected. Deactivation is accomplished by deleting the internal token identification number, after which none of the authentication steps required for user login will execute. A token is reactivated when a security officer installs a new token identification number. All prior user data is retained when a token is deactivated, avoiding the problem of rebuilding this information when the token is reactivated. The conditions which cause a token to deactivate itself are as follows:

1. Three failed login attempts. The token maintains a failure log, which is incremented each time a login fails.

2. Token expiration date is reached. The token contains an expiration date, which is compared to the current date at the beginning of each login session.

Figure 3.
Three-Way Handshake



V. KEY MANAGEMENT

In the TBACS system a user has a separate DES key for each computer on which the user is permitted access. When a user first wishes to enroll on a TBACS computer, the user must contact the computer's security officer. The security officer initializes a blank token by loading the security officer ID encrypted using a security officer PIN, the token expiration date, the user ID encrypted using an initial user PIN, and a token identification number. After receiving the token from the security officer, the token user may reset the PIN to a new value by supplying the current PIN value.

The security officer initiates a process which generates a DES key and stores the key on the token encrypted using the user's PIN and indexed by computer's identification. The DES key is also stored in the computer's key database indexed by the user's identity. This key database replaces the password database currently used on most computers.

The user may now enroll on another TBACS computer by contacting the appropriate computer security officer. As previously described, the security officer initiates a process which generates a DES key and stores the key in the token and in the computer's key database. The TBACS token is designed so that only the security officer who first initialized the token can delete token keys. Other security officers can only append keys to the token key table.

In some situations it may be desirable to eliminate the key database stored in the computer. One possible method for accomplishing this task is to assign a single master key to the computer. This master key can be easily stored in the host computer's encryption module for extra security. DES keys for user tokens are generated from the master key by encrypting the user ID using the master key. Whenever the user attempts to login the user DES key is regenerated by again encrypting the user ID using the master key. Thus, only a single secret master key needs to be maintained by the computer or its encryption module.

VI. OTHER CAPABILITIES

A. Random Key Generation

The primary purpose of the token is to generate random challenges and to perform the encryption of challenges as part of the three-way handshake used in the authentication process. However, the token can be used as a portable key generator. The token can be commanded to generate a 64-bit random number which may be used to derive a DES key by the workstation or host cryptographic module.

B. Encryption

The token can also be used for data encryption. Both the Electronic Codebook and the Cipher Block Chaining modes are supported [10]. The communications overhead required to pass the data between the reader/writer and the token along with the overhead of the algorithm may make encryption of large amounts of data impractical. Nevertheless, it may be feasible to encrypt human interactive terminal to host communications. The token can also be used as part of an automated key distribution system to decrypt new cryptographic keys sent from the host.

C. MAC Calculation

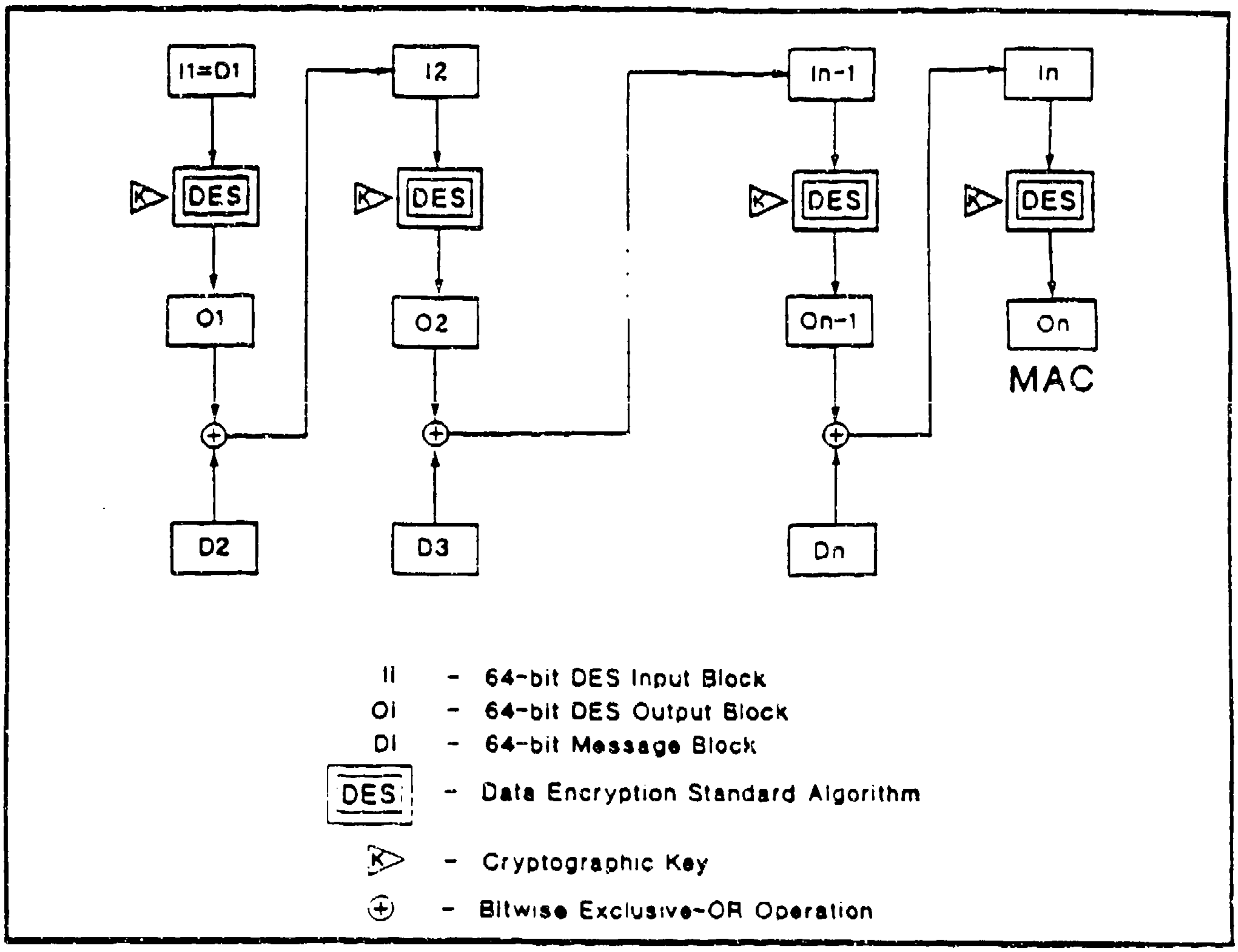
The token may be used to detect unauthorized modifications to messages by calculating a Message Authentication Code (MAC) as defined in ANSI X9.9 [6]. This algorithm is currently being used to authenticate Electronic Funds Transfer (EFT) messages worth trillions of dollars each day. The MAC computation is similar to Cipher Block Chaining encryption except that the MAC is selected from the last cipher block (Figure 4). The unencrypted data and the MAC are transmitted to the receiver. The receiver performs the MAC computation on the received message and compares the computed MAC to the received MAC. If the two values are equal then the message is accepted as unmodified. If the two values are not equal an unauthorized modification is assumed. As with data encryption, MAC computations on large messages may prove time consuming using the token. However, a message digest algorithm may be used to reduce a large message to a few 64-bit blocks which are then MACed by the token.

D. User Authorization Code Storage

The TBACS token can store user authorization codes which may control user access to information in the workstation or host computers. These codes can be passwords or read/write permissions for specific files or categories of files. A code may also indicate the security level of the user to help enforce mandatory access controls. The possible benefits of storing access control information in a token rather than in the target computer is a topic for future study.

Figure 4.

ANSI X9.9 DES Based MAC Calculation


I1=D1
I2
In-1
In
K
DES
O1
O2
On-1
On
MAC
D2
D3
Dn
Ii - 64-bit DES Input Block
Oi - 64-bit DES Output Block
Di - 64-bit Message Block
DES - Data Encryption Standard Algorithm
K - Cryptographic Key
⊕ - Bitwise Exclusive-OR Operation

VII. CONCLUSION

Smart tokens can play a major role in solving access control and other security problems. The computational capability of smart tokens can be used to perform cryptographic functions to authenticate users and protect data from disclosure and modification. Smart tokens permit cryptographic security mechanisms to be moved closer to the user where they may be protected by the user. Smart tokens can also provide conveniences for the user which make improved security requirements acceptable.

REFERENCES

1. Beardsley, Charles W., Is Your Computer Insecure? IEEE Spectrum, IEEE, Inc., New York, NY, January 1972, pp. 67-68.

2. Walker, Burce J., and Ian F. Blake, Computer Security Protection Structures, Dowden, Hutchinson and Ross, Inc., 1977.

3. Password Usage, National Institute of Standards and Technology (U.S.), Federal Information Processing Standards Publication 112, National Technical Information Service, Springfield, VA, May 1985.

4. Data Encryption Standard (DES), National Bureau of Standards (U.S.), Federal Information Processing Standards Publication 46, National Technical Information Service, Springfield, VA, April 1977.

5. Computer Data Authentication, National Institute of Standards and Technology (U.S.), Federal Information Processing Standards Publication 113, National Technical Information Service, Springfield, VA, May 1985.

6. American National Standard for Financial Institution Message Authentication (Wholesale), ANSI X9.9-1986, American Bankers Association, Washington, DC.

7. American National Standard for Financial Institution Message Authentication (Retail), ANSI X9.19-1985, American Bankers Association, Washington, DC.

8. Draft American National Standard for Financial Institution Sign-On Authentication for Wholesale Financial Systems, ANSI X9.26-198x, Draft 6.1, American Bankers Association, Washington, DC.

9. Smart Card Technology: New Methods for Computer Access Control, National Institute of Standards and Technology, Special Publication 500-157, National Technical Information Service, Springfield, VA, September 1988.

10. DES Modes of Operation, National Bureau of Standards (U.S.), Federal Information Processing Standards Publication 81, National Technical Information Service, Springfield, VA, December 1980.

APPENDIX A: TOKEN COMMAND SET

1) COMMAND: 00- RESET

INPUTS: NONE

PURPOSE: To allow for recovery from a critical error by resetting the token's temporary global variables to their initial state at power-on. The values stored in non-volatile memory are not affected.

2) COMMAND: 03- Enter SO PIN

INPUTS: SO PIN, SO ID, Token expiration date

PURPOSE: This command allows an SO to initialize a blank token by entering the required input parameters. After this command has been executed, only this SO will be able to enter the user PIN, null a value in the key table, or reactivate a token.

3) COMMAND: 04- Authenticate SO

INPUTS: SO PIN, SO ID

PURPOSE: To authenticate the SO by matching the input parameters against those stored on the token. Flag F2 is set upon successful completion.

4) COMMAND: 05- Enter User PIN

INPUTS: Old User PIN, New User PIN, User ID

PURPOSE: Allows SO to enter User PIN onto the token. The ID is encrypted under the PIN and then stored. This command can also be executed by the user in order to change the value previously stored on the token.

5) COMMAND: 06- Load Key

INPUTS: Host ID, Key, User PIN

PURPOSE: Allows an SO to load a host ID and corresponding key onto the token, granting the user access to that host. The token encrypts the key under the user PIN and stores the resulting value.

6) COMMAND: 07- Authenticate Token

INPUTS: Workstation ID, Random Number (RN1), date (YYYYMMDD)

OUTPUTS: Token PIN

PURPOSE: To verify the authenticity of the token to the user. The workstation displays the TIN to the user for verification.

7) COMMAND: 08- Generate Challenge

INPUTS: Workstation ID

OUTPUTS: Random Number (RN1)

PURPOSE: This command is the first step of the three-way handshake authentication. The workstation ID is stored for later use in key selection, and a random number is generated, stored and transmitted back to the workstation.

8) COMMAND: 09- Authenticate User

INPUTS: eK(user PIN XOR RN1), user ID

PURPOSE: Verifies the authenticity of the user based on the user PIN and ID. The user PIN is decrypted, extracted from RN1, and then used as the key to encrypt the user ID. The resulting value is then compared to the value stored on the token.

9) COMMAND: 10- Change Token PIN

INPUTS: (old token PIN), (new token PIN), workstation ID

PURPOSE: Allows the user or SO to change the current token PIN. If the old token PIN matches the value stored on the token, the new PIN is stored.

10) COMMAND: 11- Workstation Verify and Respond

INPUTS: eK(RN1), RN2

OUTPUTS: eK(RN2)

PURPOSE: To complete the final steps of the three-way handshake between the token and the workstation. The workstation encrypts the random number (RN1) received from the previous generate challenge command and generates a second random number (RN2). These values are sent to the token as input parameters for this command, which decrypts and verifies RN1. RN2 is encrypted and sent back to the workstation, which then decrypts and verifies it. This completes the three-way handshake.

11) COMMAND: 12- Output ID Table

INPUTS: none

OUTPUTS: Data block containing host IDs from key table

PURPOSE: Transfers the token's table of host IDs to the workstation, which uses this information to display a menu of available hosts to the user. Since the ID table may be larger than the capacity of the buffer, this command returns a NACK each time it is executed until the entire ID table has been transferred, at which time an ACK is returned. The workstation software checks this return value and repeatedly executes this command until an ACK is transmitted.

12) COMMAND: 13- Host Verify and Respond

INPUTS: eK(RN1), RN2

OUTPUTS: eK(RN2)

PURPOSE: Completes the three-way handshake process between the token and a remote host. This command is analogous to the workstation verify and respond.

13) COMMAND: 14- Read Zone

INPUTS: zone name

OUTPUTS: Contents of the specified zone

PURPOSE: To access the contents of a memory zone.

TABLE OF PERMISSIONS FOR ZONE COMMANDS

ACCESS TYPE:

ZONE	READ	WRITE	APPEND
0	all	user	user
1	user	none	SO

14) COMMAND: 15- Write Zone

INPUTS: zone name, data block

PURPOSE: To transfer data to a given memory zone on the token.

15) COMMAND: 16- Append Zone

INPUTS: zone name, data block

PURPOSE: To append data to a given memory zone on the token.

16) COMMAND: 17- CALLDES

INPUTS: 2-byte mode selector:

Bit 0 - set new key
Bit 1 - encrypt/decrypt
Bit 2 - load B from input buffer
Bit 3 - xor two input values (A ^ B)
Bit 4 - produce output

16-byte key or padding(required)
16-byte ASCII hex data string A
16-byte ASCII hex data string B (optional)

OUTPUTS: NACK or ACK and 16-byte result, unless output is suppressed (bit 4 of mode byte is 0).

17) COMMAND: 19- TEST

NOTE: The inputs consist of a 1-byte mode selector and additional parameters which are depe.,dent on the mode selected, as follows:

	MODE: 0	 1
INPUTS:	data	none
OUTPUTS:	data	f_log) "NULL"

PURPOSE: This command provides the following test modes:

0- Echo data
1- Current token status

THE BOEING MLS LAN:
HEADED TOWARDS AN INFOSEC SECURITY SOLUTION

Gary R. Stoneburner and Dean A. Snow

Boeing Aerospace and Electronics
P.O. Box 3999, MS 87-06
Seattle, WA 98124-2499

Introduction

This paper describes how and why the Boeing Multilevel Secure Local Area Network (MLS LAN) is migrating towards an Information Security (INFOSEC) solution for providing protection against many of the security threats facing Local Area Networks (LANs) today. INFOSEC is a combination of Computer Security (COMPUSEC) and Communications Security (COMSEC). We are investigating the addition of an encryption capability in to MLS LAN. This will complement an already existing set of security mechanisms which have been designed and built to satisfy the class A1 set of requirements for COMPUSEC as specified by the Trusted Network Interpretation (TNI) of the DOD Trusted Computer System Evaluation Criteria (TCSEC) [1]. This paper includes a description of the MLS LAN history and why the addition of encryption is a desirable option for many applications. It will present the significant design issues and give a preliminary overview of how encryption might be embedded into the MLS LAN.

Background and Overview of MLS LAN

In 1983, an internal research and development group within Boeing Aerospace and Electronics, a division of the Boeing Company, began developing the high performance, fiber optic based MLS LAN. The purpose of the Boeing MLS LAN is to allow users at different security levels to simultaneously process multiple levels of data on the network. Both single-level and multilevel subscriber devices operating at different security levels can be attached to the network. The MLS LAN guarantees the separation of user data at different security levels and provides access controls regulating the access of the users to the network devices and data. The MLS LAN is targeted for advanced C3I applications supporting airborne, mobile ground, and fixed ground sites. For an example of how the MLS LAN would fit into a very large campus installation, such as an Army base, refer to [2].

Figure 1 shows the MLS LAN system and illustrates the extensive services it provides for its users. These services include -

Network terminal access,

Terminal-to-terminal communications,

Terminal-to-host communications,

Host-to-host communications,

Video circuit switching control, and

High-speed digital stream circuit switching control.

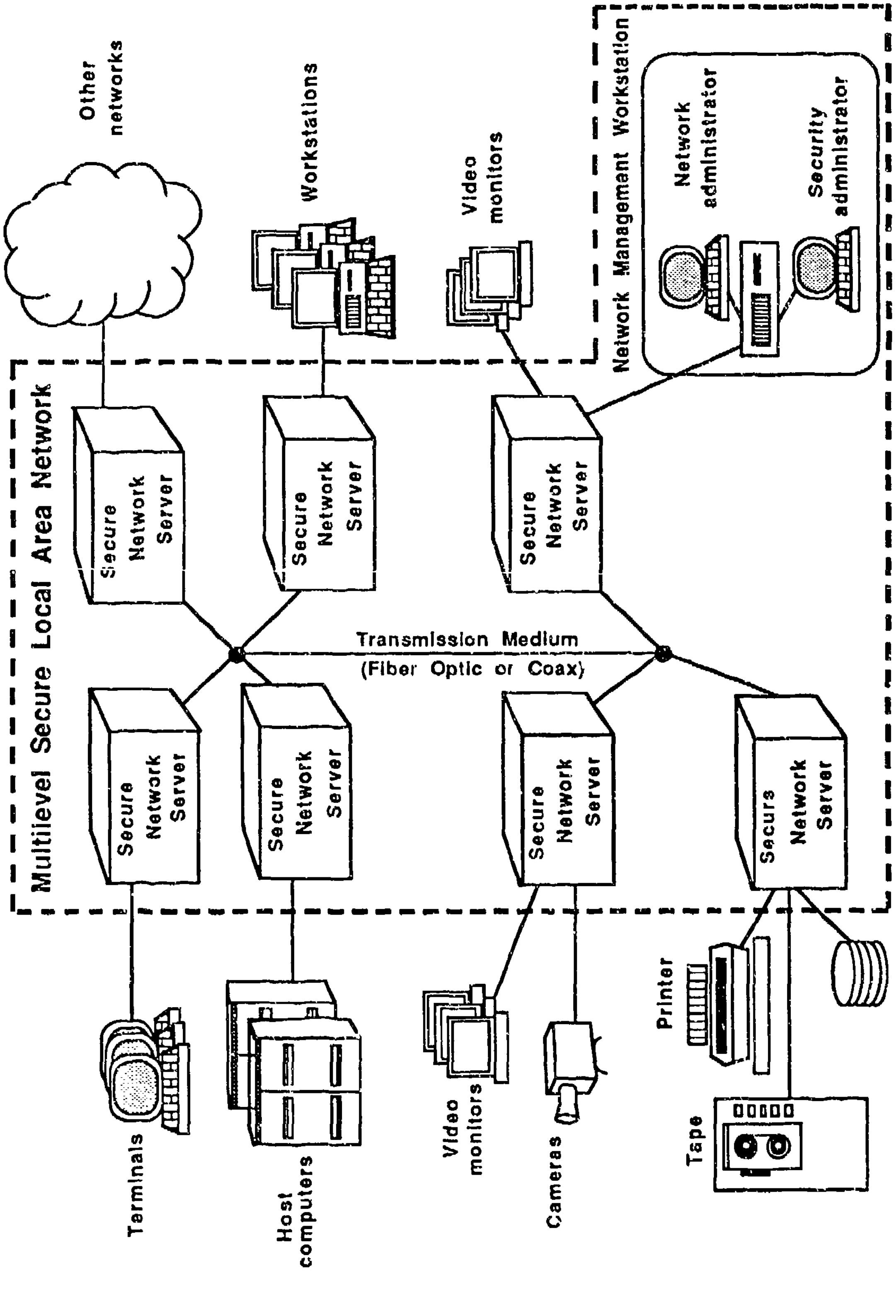


Figure 1. MLS LAN System Diagram

Major Components of MLS LAN

As shown in Figure 1 above, the MLS LAN is composed of three major components, the Secure Network Servers (SNS), the fiber optic (or coaxial) transmission medium, and the Network Management (NM) workstation. These components are described below.

Secure Network Servers (SNS)

The MLS LAN SNSs contain interfaces to user devices and an interface to the transmission medium. Depending on the configuration chosen, the SNSs provide an interface to a fiber optic or a coaxial transmission medium. User device interfaces (hardware and software) developed thus far include support for both single-level and multilevel user terminals, workstations, host computers, video devices, and digital stream devices (e.g., optical disks). The modular design of the SNSs allows for easy development and addition of new user interfaces.

The DOD protocol suite is implemented within each SNS to route the data across the network. Protocol support is included for TELNET, Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Protocol (IP). Future MLS LAN plans include support for the ISO/OSI protocol suite.

MLS LAN is a broadcast network. When an SNS transmits a packet of data it is broadcast to all SNSs on the network. Each SNS checks every packet for its own address. If an SNS recognizes the address as its own, the packet is routed to the appropriate attached subscriber device.

Most of the network multilevel security mechanisms are built into the hardware and software resident within each SNS. The totality of the network security mechanisms (including that portion which is implemented in the network management workstation) is referred to as the network trusted computing base (NTCB).

Transmission Medium

The second major component of the MLS LAN is the transmission medium. MLS LAN can be configured with either a fiber optic trunk or an Ethernet coaxial trunk. The fiber optic trunk consists of the fiber optic cables and passive star couplers used to connect the SNSs. Wavelength division multiplexing is used to simultaneously support the transmission of digital (100 Mbps), analog video, and high-speed digital stream (275 Mbps) data across the fiber optic trunk. The IEEE 802.4 token bus protocol is used to gain access to and route digital multiplexed data across the fiber optic trunk. The MLS LAN system architecture is based on a fragmented star topology. Multiple SNSs are connected to star couplers by the fiber optic cables. Fragmented stars are formed by interconnecting the star couplers. The MLS LAN bus topology requires that every star coupler be connected to every other star coupler within the network. When an SNS or cluster of SNSs connected to the same star coupler fails, the remaining SNSs are able to continue communicating.

As an option to its fiber optic configuration, MLS LAN offers an Ethernet communications trunk. The Ethernet option uses the IEEE 802.3 protocol, and coaxial cables to route the data across the communications trunk. The video and digital stream services are not available with the Ethernet option.

Network Management (NM) Workstation

The third major component of the MLS LAN is the NM workstation. The NM workstation is connected to a single SNS via a special NM interface which is a standard feature of all SNSs. The NM workstation contains the network and security administrator interfaces to the network. The network administrator is responsible for network configuration and performance monitoring. The security administrator is responsible for setting the security parameters for the network users and devices, and for monitoring the network audit events and alarms collected and stored at the NM workstation. The NM workstation is a part of the NTCB.

MLS LAN COMPUSEC Evaluation

The MLS LAN has been developed and designed to satisfy the A1 requirements of the National Computer Security Center's (NCSC) TCSEC. In July of 1987, the NCSC released a new set of requirements and criteria tailored towards trusted computer and communications network systems. This document is known as the Trusted Network Interpretations (TNI) of the TCSEC [1].

MLS LAN has been in the NCSC COMPUSEC evaluation program since 1985. MLS LAN (less the NM workstation) has now entered formal evaluation with the NCSC. The NM Workstation is in developmental evaluation and will enter formal evaluation as part of the MLS LAN at a later date.

Encryption vs. Physical Protection Requirements

Although the MLS LAN has been designed to satisfy the requirements and meet the criteria of the TCSEC, without a data confidentiality service its transmission media and SNSs and NM must be physically protected to system high to protect against passive and active wiretapping attacks from both external and internal threats.

The MLS LAN Trusted Facility Manual (TFM) outlines the security design concept and defines the guidelines for trusted facility management for the MLS LAN. It assigns to the network and security administrators the responsibility for physically securing the MLS LAN. The TFM requires that the MLS LAN transmission media, star couplers, SNSs and network management workstation be protected by measures commensurate with the highest level of data processed by the network. It also requires that the subscriber devices and their device interfaces be protected to the maximum level of data processed by or stored in the device.

For many applications, the physical protection of the network equipment is an easy requirement to meet. However, for other applications, the cost or feasibility of providing secure facilities to house the SNSs and providing protected wire-line distribution systems to house and protect the network transmission media preclude this solution and an alternative method for protecting the network data is required. Properly applied encryption techniques negate the physical protection requirements for the transmission media and also reduce the physical protection requirements for the SNSs from system high protection to measures appropriate for the highest level of plain text data or cryptovariable processed by the SNS.

Encryption and the TNI Part II Security Services

The TNI "provides interpretations of the Department of Defense (DOD) TCSEC for trusted computer/communications network systems" [1] and is divided into two parts:

Part I of the TNI interprets the TCSEC requirements for networks, including LANs such as the Boeing MLS LAN.

Part II of the TNI describes additional security services for network applications and describes qualitative ratings to be assigned to networks. The additional security services described in part II are divided into three major categories; and each of these major categories are further broken down into three subcategories for a total of nine subcategories. High ratings are anticipated for each of the security services MLS LAN provides, as long as the SNSs, NM workstation, and transmission medium are physically protected. The addition of an encryption capability will allow MLS LAN to retain its security service ratings without the caveat of physical protection. For a detailed description of how MLS LAN currently provides for these nine security services see [3].

The three major categories of additional security services described in part II are communications integrity, denial of service, and compromise protection.

Communications Integrity

Communications integrity is divided into the security services of authentication, communications field integrity, and non-repudiation. The application to MLS LAN is described below.

Authentication: Authentication provides an assurance as to the identity of a communications entity. It provides protection against a subversive entity masquerading as a legitimate network entity and against the replay of previous network traffic. Currently MLS LAN does not provide any means to directly authenticate its communicating entities. MLS LAN relies on the physical security of its network components, and detects and reports the disconnection of network components to provide a reasonable assurance as to their identity.

Communications field integrity: Communications field integrity provides protection against the unauthorized modification of network traffic. This service protects against Message Stream Modification caused by active wiretapping. Currently MLS LAN relies on NTCB integrity mechanisms such as Cyclic Redundancy Codes, checksums, error-detecting memory, and non-NTCB TCP integrity mechanisms such as checksums, and packet sequencing to provide for communications field integrity. Current communications field integrity mechanisms rely on the physical protection of the MLS LAN components to protect against active wiretapping.

Non-repudiation services: Non-repudiation services protect against after-the-fact denial of message transmission or receipt. The MLS LAN does not presently provide non-repudiation services.

The addition of encryption into MLS LAN could strengthen its communications integrity services. Encryption services provide for peer entity authentication, and provide cryptologic checksums and data protection mechanisms to detect MSM.

Denial Of Service (DOS)

DOS security services provide protection against the unauthorized or inadvertent denial of network access and resources to human users and subscriber devices and detect or prevent conditions that cause a reduction in network throughput below an established minimum value. MLS LAN uses traditional methods to address DOS; including control of resource utilization, detection of network component failures, and the collection of performance data to detect reduced throughput. The strength of MLS LAN's resistance to DOS is dependent upon the physical protection of the network components. Encryption can play a role in DOS by protecting the mechanisms used to address DOS against active wiretapping (unauthorized modification) when physical protection is not provided. This includes protecting against and detecting the replay of a previous communications session which could block network access and consume network resources.

Compromise Protection

The TNI subdivides compromise protection into the services of data confidentiality, traffic flow confidentiality, and selective routing. Data confidentiality is the protection of network traffic from unauthorized disclosure via passive wiretapping. Traffic flow confidentiality protects against traffic analysis, which is gleaning information from network traffic other than data, such as message length, frequency, timing, and addresses. Selective routing is the capability to route traffic through a more desirable communications link, avoiding specific links which may have been subverted or through which certain information is restricted. Each of these items is discussed below in the context of the MLS LAN.

Data confidentiality: Currently MLS LAN requires the physical protection of its network components (SNSs, fiber optic trunk and NM workstation) to protect the network traffic from unauthorized disclosure, assuming no data confidentiality service. Adding encryption to MLS LAN will provide a strong data confidentiality service and a limited traffic flow confidentiality service. The strength of the data confidentiality service dependents on the strength of the encryption algorithm used, the granularity of the encryption keys, and the method of embedding the encryption devices into the network.

Traffic flow confidentiality: The strength of the traffic flow confidentiality service depends on the layer in which the encryption service is provided (the lower the layer the stronger the service will be) and on use of additional traffic padding services.

Selective routing service: Since MLS LAN is a broadcast network, it does not provide selective routing; however, a future MLS LAN gateway node will support this service.

The NSA Commercial COMSEC Endorsement Program

It was for the above reasons that the Boeing MLS LAN applied for admittance into NSA's Commercial COMSEC Endorsement Program (CCEP). A Memorandum Of Understanding (MOU) was signed by NSA and Boeing Aerospace and Electronics in August, 1988.

NSA, through the CCEP, provides industry with COMSEC technical expertise and defines security requirements for proposed telecommunications systems. Industry then develops the new secure communications systems using NSA-proprietary, classified cryptography. Once the new system has been certified as meeting the security requirements, NSA places the system on the Endorsed Cryptographic Products List. Industry is then free to manufacture and market the endorsed system for use in securing government information.

Goal of MLS LAN Participation In CCEP

The goal of MLS LAN participation in the CCEP is to embed the COMSEC modules into the SNSs to -

Provide an encryption capability transparent to network users and attached subscriber devices;

Receive an endorsement from NSA to secure the full range of classified information (Type 1);

Negate the physical protection requirement for the transmission medium and reduce the physical protection requirement for the SNSs;

Maintain, in more hostile environments. the ratings assigned to MLS LAN for the security services it provides, as specified in Part II of the TNI;

Minimize the changes required to the current SNS architecture;

Be compatible with the proposed ISO security architecture;

Be compatible with Secure Data Network System (SDNS) protocol; and

Maintain the high data rate of the fiber optic trunk.

Encryption Design Issues

There are a number of issues to be considered and design decisions to be made prior to adding an encryption capability into an existing system. Major issues include the type of encryption to be used in the system; the key management techniques providing the accounting, distribution and control of the encryption keys; and the location where the cryptographic module will be embedded within the system.

Network Encryption Modes

The two most common modes of encryption are link level and end-to-end. Each has its own distinct advantages and disadvantages which are briefly described below. End-to-end encryption appears best suited to the encryption goals of MLS LAN.

Link Level Encryption (LLE): The purpose of link level encryption (LLE) is to protect data as it traverses the most vulnerable part of a communications network, the transmission media. LLE is characterized by the encryption taking place, according to the ISO reference model, at the link layer or below so that everything above the link layer is encrypted.

LLE is normally accomplished by attaching encryption devices to opposite ends of a communications line and external to the network nodes. However, in some cases the LLE devices are embedded within the communication equipment (or network nodes). In this case, the encryption can take place at either the bottom or top of the link layer.

Encryption taking place at the bottom of the link layer encrypts the entire message and maintains the strong traffic flow security described above. A major drawback to encryption taking place at the bottom of the link layer is that messages are encrypted as they exit, and decrypted as they enter network nodes. This means that messages not intended for a network node are vulnerable to attack while within that node. This is contrary to the MLS LAN goal of reducing the physical protection requirement of the SNSs.

Placing the encryption at the top of the link layer solves this problem, as it allows the node to recognize its own unencrypted network address and only decrypt the data portion of messages intended for it. Thus, data not intended for a specific node (or SNS) remains protected (encrypted) while within that node. However, since the link layer protocol information is not encrypted, traffic flow security is reduced and traffic padding and masking techniques are required to protect against traffic analysis.

LLE provides strong traffic flow security as entire messages, including the headers themselves, are encrypted. In addition, LLE key management techniques are relatively simple. Normally, each unique communications link is protected by a different key, minimizing the number of keys used in the network. A drawback to the simplified key management is that fewer keys put the network at greater risk for compromise.

End-to-End Encryption (E3): The primary goal of end-to-end encryption (E3) is to protect the flow of data between two communicating entities (such as two network processes) over their entire communications path.

E3 is characterized by an encryption scheme that encrypts all of the user data but leaves a portion of the message header in the clear. Therefore, the data portion of messages need not appear in clear text form except at the originating and destination nodes; which is consistent with the MLS LAN goal of reducing the physical protection requirement for the SNSs.

E3 can be based on communication sessions. In this case, a single session key is negotiated between the two communicating devices, is used to encrypt the data for the life of the session, and is destroyed once the session has been terminated. Session keys provide excellent separation between users.

E3 usually takes place at either the top of the network layer or the bottom of the transport layer. All protocol control information in the layers below the point of encryption remain in clear text. In this regard, a decision must be made as to where to place the encryption.

By using the network nodes as the E3 end points, only the network hardware and software itself must interface to the E3 security mechanisms. Also, encryption transparency is provided to both the network users and attached subscriber devices. This has the drawback that the communication lines between attached subscriber devices and network nodes must be physically protected, but provides the advantage of reducing the scope of the complex E3 interface. Where it is not practical to physically protect the communication lines between subscriber devices and network nodes; external, line-level encryption devices would provide the necessary protection.

The main disadvantage of E3 is that with more keys to handle, key management is more complex. Also, if traffic flow security is required, then traffic masking and padding techniques must be used because message headers are unencrypted.

Key Management Issues

Key Distribution: There are a number of conventional and emerging key distribution technologies being evaluated for use within MLS LAN.

One conventional technique is manual key distribution, where keys are manually downloaded into a cryptographic module through a key fill device. This method can be time consuming and error prone. Another conventional method is a key distribution center, where keys are centrally located and automatically distributed as needed. Centralized key storage and distribution can lead to a single point of failure.

An emerging key management technique currently being promoted by the SDNS initiative is a derivative of public key encryption [4]. This keying system is based on a two step process. Initially, an external key management center issues startup keys. Then, when two entities desire to communicate, they exchange keying information and generate a unique pair-wise traffic key. This emerging technology is well-suited to MLS LAN's session-oriented type of traffic.

Key Granularity: Cryptographic keys are used in varying granularity.

Network keys are a form of key where one key is used to encrypt and decrypt all traffic on the entire network.

Node keys are a form of keying in which each network node has its own unique key. With node keys, an entity desiring to communicate with another entity on a different node must have access to the destination node's key and then use it to encrypt the message.

Session keys are based on communications sessions. Session keys are used for all or part of a single communications session and then destroyed.

Keys based on security labels are another common form of keying. This form of keying uses a separate key for each security level or security range and can even be used to provide separate keys for individual security compartments. Keys based on sensitivity labels provide good separation between data of different sensitivities.

Key Granularity as Related to MLS LAN Traffic Types: The different types of network traffic on the MLS LAN lend themselves to different types of keys. Traffic relating to the management, maintenance, and security reporting of the network is well suited to a form of net key. This type of traffic, (including network startup and shutdown messages, network address translation messages, user authentication requests and responses, and security auditing messages) always involves communications between the NM SNS and one or more of the remote SNSs. A slightly more complex and more secure keying scheme for these types of messages would be to use node or SNS keys, in which case each SNS (node) would be assigned a different key. With this keying scheme the NM SNS would have to store multiple keys, one for each of the remote SNSs. Each remote SNS would have to store the key of the NM SNS.

Another form of traffic on the MLS LAN is connection-less traffic between SNSs and between subscriber devices. Examples of this type of traffic are inter-terminal messages, host-to-host UDP messages, and TELNET session establishment and termination messages. This type of traffic, like the previously described traffic, is well-suited for a SNS (i.e., node) keying scheme. A variation of the SNS keying scheme would be to use a different key for each security classification of data within the security range of the SNS.

The third type of traffic on the MLS LAN is connection-oriented traffic between SNS subscriber devices such as terminals and host computers. Examples of this form of traffic are terminal-to-host TELNET sessions and host-to-host TCP connections. This type of traffic lends itself to dynamically created keys that are created and destroyed when the connections are established and terminated respectively. Dynamic keying schemes, such as the SDNS public key exchange, will be investigated to determine their applicability to MLS LAN.

Cryptographic Architecture

One of the major decisions facing a system designer embedding cryptographic modules is where to place the modules(s) within the network. A primary goal while embedding encryption into MLS LAN will be to minimize the changes that will be required to the existing hardware and software. This must be accomplished while maintaining red/black separation which is keeping unencrypted (red) data isolated from areas where encrypted (black) data is processed.

Approaches to Embedded Cryptography

Three basic concepts for embedding cryptographic modules into a distributed processing system are identified in [5]. In one of these approaches, the dedicated cryptographic communication subsystem, the cryptographic module resides within the end-user system, such as a host computer, and is outside the scope of a LAN. The MLS LAN solution will be a combination of the other two approaches; namely cryptographic coprocessor and split-bus. These two approaches are briefly described below.

In the cryptographic coprocessor approach, the cryptographic module would be attached directly to the system bus.

With the split-bus cryptographic approach, the cryptographic module would be embedded to split the system bus into separate red and black busses.

The MLS LAN Cryptographic Approach

Figure 2 below illustrates the current architecture of an MLS LAN Secure Network Server (SNS) . A common system bus is used to transfer all control information and to transfer all data between system memory and attached user devices. A second bus is used for all data transferred to or from the network (i.e., between system memory and the Network Frontend).

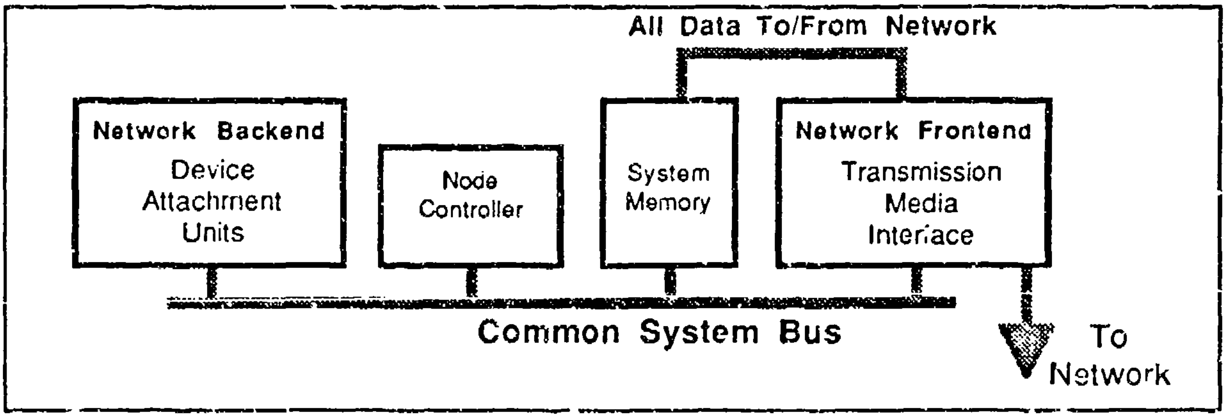


Figure 2. Current MLS LAN Architecture

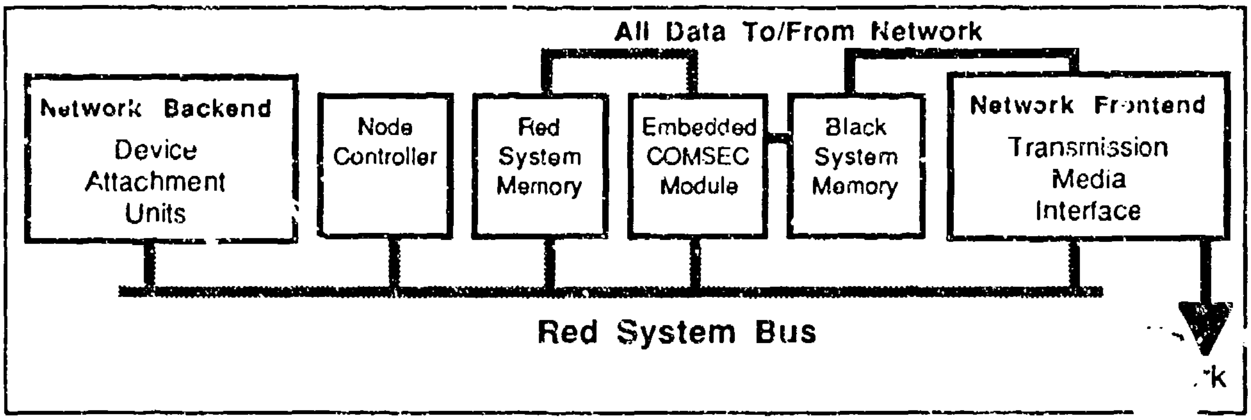


Figure 3. Proposed MLS LAN Encryption Architecture

Figure 3 above shows the proposed placement of the cryptographic processor within an SNS. This solution has been selected since it minimizes the impact to the present MLS LAN architecture. The red/black data separation is provided by the existing dual, data busses. With this configuration the trusted software resident within an SNS would be responsible for initiating the encryption and decryption of data. The trusted software would also be responsible for controlling bypass of message headers; i.e., deciding which portion of a message to encrypt and which part would remain clear text. The primary problems which must be addressed are -

Definition of the COMSEC boundary. Essentially this is determining the location of all trusted software controlling the COMSEC function. The possible locations are (1) on the Node Controller and (2) on the COMSEC module itself.

Red/black separation. The major concerns are the connection of the COMSEC Module and Network Frontend to the system bus and TEMPEST concerns related to a common system bus.

Security Fault Analysis. Here the concern relates to the effects of hardware faults in existing circuits cards that share the system bus with the COMSEC module.

This proposed MLS LAN approach simplifies the red/black separation issue with the cryptographic module providing the separation between red and black data. Moreover, the approach requires no hardware modifications to existing MLS LAN fiber optic systems and only minimal changes to Ethernet systems.

This approach to MLS LAN encryption is being investigated to determine what additional assurances or mechanisms are required to verify that unencrypted data cannot be accidently or subversively routed pass the cryptographic module and out onto the MLS LAN communications trunk.

Summary

By embedding an encryption capability into its MLS LAN, Boeing Aerospace and Electronics is responding to a requirement for encryption in certain C3I communications systems. It is also responding to NSA's goal of using INFOSEC countermeasures to secure future communications and information processing systems.

References

[1] National Computer Security Center, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-005 Version-1, 31 July 1987.

[2] Philip C. Stover, "Designing Multilevel Secure Networks," presented at U.S. Army ISEC Technology Strategies '87, February, 1987.

[3] Daniel Schnackenberg, "Applying the Orange Book to an MLS LAN," in Proceedings of the 10th National Computer Security Conference, September, 1987, pp. 51-55.

[4] Ruth Nelson, "SDNS Services and Architecture," in Proceedings of the 10th National Computer Security Conference, September, 1987, pp. 153-157.

[5] John Jacobs, Thomas Kibalo, "Secure Data Network System Support using Embedded Cryptography," in Proceedings of Second Annual AFCEA Intelligence Symposium, September 1987.

Abbreviations

A1	A security level defined in the TCSEC
C3I	Command, Control, Communications and Intelligence
CCEP	Commercial COMSEC Endorsement Program
COMPUSEC	Computer Security
COMSEC	Communications Security
CRC	Cyclic Redundancy Code
DOD	Department of Defense
DOS	Denial of Service
E3	End-to-End Encryption
IEEE	Institute of Electrical and Electronic Engineers
INFOSEC	Information Security
IP	Internet Protocol
ISO	International Standards Organization
LAN	Local Area Network
LLE	Link Level Encryption
Mbps	Million Bits per Second
MLS LAN	Multi-Level Secure Local Area Network
MOU	Memorandum of Understanding
NCSC	National Computer Security Center
NM	Network Manager
NSA	National Security Agency
NTCB	Network Trusted Computing Base
OSI	Open Systems Interconnect
SDNS	Secure Data Network System
SNS	Secure Network Server
TCP	Transmission Control Protocol
TCSEC	Trusted Computer System Evaluation Criteria
TELNET	Telecommunications Network (a terminal protocol)
TFM	Trusted Facility Manual
TNI	Trusted Network Interpretation (of the TCSEC)
UDP	User Datagram Protocol

THE SILS MODEL FOR LAN SECURITY

L. Kirk Barker
IEEE 802.10 Editor, Datotek, 3801 Realty Road, Dallas TX 75010, (214) 241-4491

Kimberly Kirkpatrick
IEEE 802.10 Chair, MITRE, MS K304, Burlington Rd., Bedford MA 01730, (617)271-7555

Abstract

This paper describes the model on which the Institute for Electrical and Electronics Engineers (IEEE) 802.10 is basing its security protocols and services for Local Area Networks (LANs). The Standard for Interoperable LAN Security (SILS) will provide a standard protocol for protecting LAN traffic. It also will specify methods of key management and system/security management with supporting protocols. Currently, the Secure Data Exchange Protocol is nearing completion, and the Key Management and System/Security Management work has begun.

The Secure Data Exchange Protocol will provide such services as Confidentiality, Integrity, and Access Control for data. It also will allow the provisioning of such mechanisms as encryption and security labels. The Key Management will provide for both public and private key methods, and the System/Security Management will be compatible with the current International Standards Organization (ISO) work.

1.0 Introduction

The use of LANs and data networks in general has become wide-spread. LANs are used to transfer vast amounts of information on which both industry and Government rely to perform their daily operations. In many cases, disclosure of this information to competitors, or other governments, would severely undermine the effectiveness of the organization.

The Standard for Interoperable LAN Security is the work item of IEEE 802.10. IEEE 802.10 is co-sponsored by the IEEE Technical Committee on Security and Privacy and IEEE 802 LAN Standards Committee which is the standards organization of the IEEE Technical Committee on Computer Communications.

The IEEE 802.10 Working Group was formed to address the urgent need to provide secure communications on LANs. The requirement for standards in this area was identified by the emergence of products which encrypt the data being transmitted between hosts on a LAN. As these products were introduced, it became clear that a greater market could be addressed if the vendors could develop a standard for the secure communications. As a result of discussions among vendors and users, a meeting was held in March 1988 to determine if there was enough interest to produce a standard. The first meeting attracted 42 vendors and users. Since that time, many vendors and users have been active in the development of the model in this paper and the supporting protocols.

The expectations of the working group are that SILS will become an IEEE standard and then be submitted to ISO through the American National Standards Institute (ANSI) for consideration as an ISO standard. Naturally, as the standard progresses, members of the appropriate ANSI and ISO bodies are kept informed as to the direction and objectives of the standard. In addition, all attempts are being made to remain compatible with the on-going security work in ANSI and ISO, especially the ANSI X3T5.4, X9, and the ISO/IEC JTC 1 SC 20 and SC 21 groups. A goal of the working group is that when the standard is presented as an IEEE standard, there will be no surprises to the ANSI or the ISO committees.

This paper presents a snapshot of the model that the IEEE 802.10 group is using to define security for LANs and as such reflects the work of the members as well as the authors. The body of this paper describes the concerns of the IEEE 802.10 effort and then the model. Details of the Secure Data Exchange Protocol, Key Management, and System/Security Management are contained in the discussion of the model. Since the SILS standard is still under development, all of the information presented in this paper is subject to change.

IEEE 802.10 is defining three areas for standardization: Secure Data Exchange (SDE), Key Management, and System/Security Management. These areas of standardization have been designed so that the use of one does not mandate the use of either of the other two interfaces. This allows specific implementations to specify compliance to SILS Key Management, SILS Secure Data Exchange, and SILS System/Security Management independently.

The Open Systems Interconnect (OSI) Basic Reference Model (IS 7498) defines a 7-layer communications model. The SDE is an OSI Layer 2 protocol that provides services to allow the secure exchange of data at Layer 2. The Key Management Protocol is a Layer 7 protocol that provides services for the management of the cryptographic keys used to encrypt the data at Layer 2. The System/Security management is a Layer 7 set of services that is used to securely manage the security protocols.

The model for SILS shows these interfaces, explains which aspects of the interfaces are defined in this standard, and shows the relation of SILS to OSI. Section 2.0 presents issues he IEEE 802.10 Working Group is addressing. Section 3 explains the OSI concepts relevant to SILS. Section 4 introduces the SILS protocol "stacks". Finally, Section 5 presents the detailed model.

2.0 Concerns

This section discusses the proposed relationship of SILS to the definition of a system security architecture, and to existing LAN devices.

2.1 Relationship to Entire Security Architecture

The procedures defined in SILS play a particular role in the development of a security architecture. The definition of a security policy is one of the first steps in providing the security architecture. The policy should state "The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes information" (The Orange Book DoD 5200.28). All facets of security should be considered in this policy (e.g., procedural, physical, legal ramifications, benefit/cost).

Based on the security policy, the system security requirements are defined and from these requirements, the features the overall security architecture should provide are derived. An example of a security policy is: "information will not be disclosed to unauthorized hosts." The system requirements, based on the threat posed by the environment, could be: "data while it is on the LAN shall be protected from passive wire taps and data addressed to a particular host on the LAN shall not be accessible by any other than the intended host." The security architecture which could meet these requirements would be an interface between the hosts and the LAN which implements the SILS Secure Data Exchange services of Confidentiality and Access Control.

SILS defines the security service interfaces and the associated protocols. These services can be chosen to satisfy the security requirements of a particular system. It is likely that the services provided by this standard will not satisfy all the system security requirements. Thus, the entities responsible for the LANs that operate in accordance with this standard need to ensure that the other appropriate security controls such as physical or procedural are in place.

Since, the protocols defined are independent of the particular key management and encryption algorithms that are used. The algorithms selected can be chosen to meet the security needs as specified in the security requirements in a cost effective manner.

2.2 Existing Networks

One of the biggest concerns of 802.10 is how the developed protocols will affect existing network configurations. To address this problem, at least one mode of each protocol must support a transparent implementation. A transparent implementation is one that meets the following criteria:

o It must be transparent to 802 devices that currently exist on the network. That is, an 802.10 device can be placed on a network without affecting the functioning of existing devices that do not implement the IEEE 802.10 security protocols.

By meeting the goal of transparency as noted above, implementations of SILS can be used to secure applications working across existing LANs. It also mandates the co-existence of protected and unprotected traffic on the same LAN. IEEE 802.10 is studying the effect of the security protocols on bridges and LAN analyzers.

3.0 OSI Structure

There are two Open System Interconnection Models that are relevant to the SILS model: the OSI Basic Reference Model and the OSI Management Model. In the OSI Basic Reference Model (IS 7498), protocol services are requested across a service interface. The upper (N+1)-layer requests a service from the lower (N)-layer (see Figure 1). The service varies depending upon what that layer is able to provide. The usefulness of this layering concept is that the upper (N+1)-layer protocol may request services of a lower (N)-layer without any knowledge of which mechanisms the (N)-layer employs to implement the service.

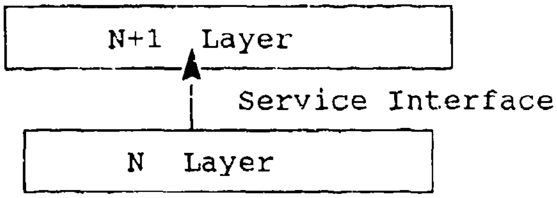


Figure 1 -- Protocol Layering

The IEEE 802 architecture maps onto this model as shown in Figure 2. The Media Access Control (MAC) layer consists of all OSI layer 1 and part of OSI layer 2. This MAC layer contains the LAN media CSMA/CD; IEEE 802.3, Token Bus; IEEE 802.4, Token Ring; IEEE 802.5, Metropolitan Area Network; IEEE 802.6. The Logical Link Control (LLC); IEEE 802.2 resides above the MAC layer and is in OSI layer 2.

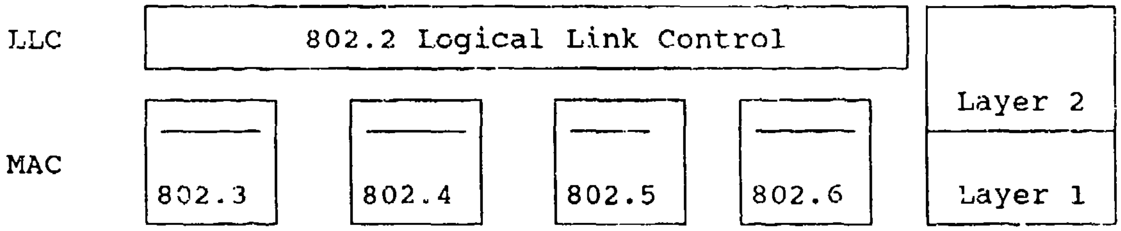


Figure 2 -- IEEE 802 Architecture

The OSI Management Model (IS 7498/4) defines management as system management and layer management. System management uses all seven OSI layers for monitoring and controlling the network. Layer Management acts directly at a single layer. The architectural entities required to manage an Open System are System Management Application Entities (SMAEs), Layer Managers (LMs) and Management Information Bases (MIBs). Figure 3 shows the placement and relation of these entities. The protocols which can be used by these management entities are the Common Management Information Protocol (CMIP IS 9495) and IEEE 802.1 Systems Management Protocol.

For each protocol at each layer, there is a Layer Manager associated with that protocol. The depiction of LMs in Figure 3 shows a LM concatenated with each layer to represent the requirement for an LM for each layer and for each protocol in each layer. The LM may communicate with other LMs at its layer (e.g loopback) to manage the layer. Usually, the main function of the LM is to manage the objects used by the protocol. The operations on the objects (such as GET or SET or generation of EVENTS) are performed by the LM as directed by the SMAE.

The communication of the LMs and the SMAEs is a local matter and is defined internally by an end system. This internal communication is shown by the inverted L-shaped boxes in Figure 3 and is often referred to as the management "cloud." The cloud consists of local implementation procedures and data that span all the layers of the model.

SMAEs are layer 7 application entities that perform the systems management of a network. They communicate via CMIP or IEEE 802.1 protocols. Thus the communication for systems management is standardized.

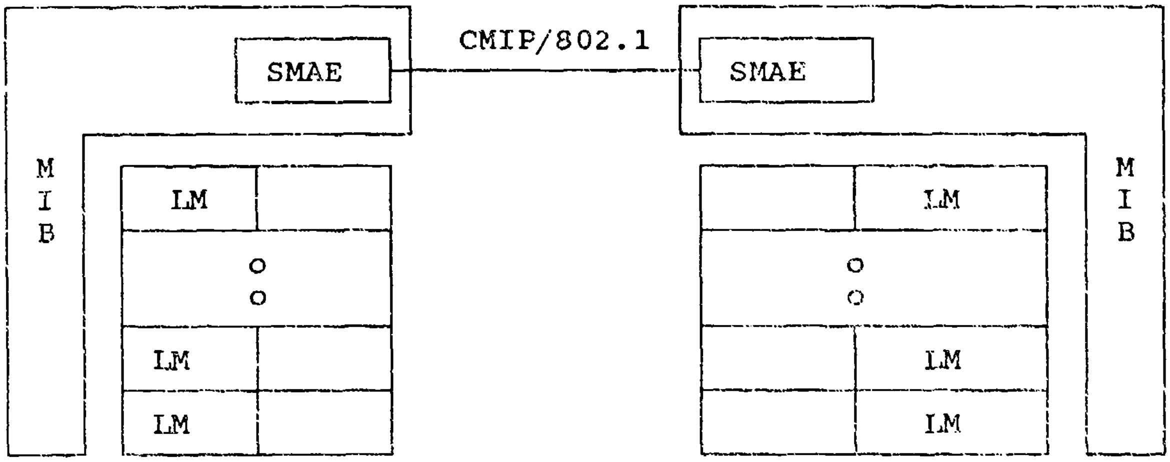


Figure 3 -- Layer Managers

The objects that must be managed are defined with respect to each protocol. Examples of these objects for communications protocols are window sizes, timers, and buffer sizes. For security protocols, some objects will be devoted to supporting the security mechanisms employed. For instance, if encryption is used, a managed object might be the cryptoperiod of a key. For security protocols, those objects that count significant events such as failures and then generate an event once the count reaches a certain threshold are extremely useful. These objects are stored in the MIB. To further depict the protection and separation of security-related objects from other management objects, the concept of a Security MIB (SMIB) is introduced. The structure of the SMIB or the MIB is a local issue; however, the structure of the objects is standardized and is defined in the Structure of Management Information (SMI IS DP 10165-2).

3.0 SILS Protocol Stacks

It is common to refer to the protocol which supports the service interface and the protocols beneath this protocol as a "stack". Using this terminology, Figure 4 shows the stacks defined for SILS: a Key Management Stack, multiple Data Exchange User Stacks, and a System/Security Management Stack.

Each stack must have a separate Layer 2 entity. In the case of System/Security Management and Key Management, this Layer 2 entity is LLC which is why Figure 4 shows multiple instantiations of LLC directly above the SDE sublayer. The only requirement of these Layer 2 entities is that they can provide the (N+1) layer interface expected by the SDE service interface.

3.1 Data Exchange User Stack

On the left in Figure 4 are boxes entitled "Data Exchange User Stack". These stacks request services from the Secure Data Exchange Service Interface. While the Key Management and System Management stacks may be additions required by the SILS protocols, these "User Stacks" are what currently exist on many Local Area Networks. The User Stack can also be thought of as a client of the SDE stack. It requests security services from the SDE service interface.

While the SDE protocol provides an interface to the Data Exchange User Stack, SILS does not specify any of the protocols that reside in this stack. The User Stack may be any protocols that would normally reside directly above the MAC layer. The most obvious of the protocols is the LLC, but it could easily be other protocols as long as the protocol maintains the MAC interface.

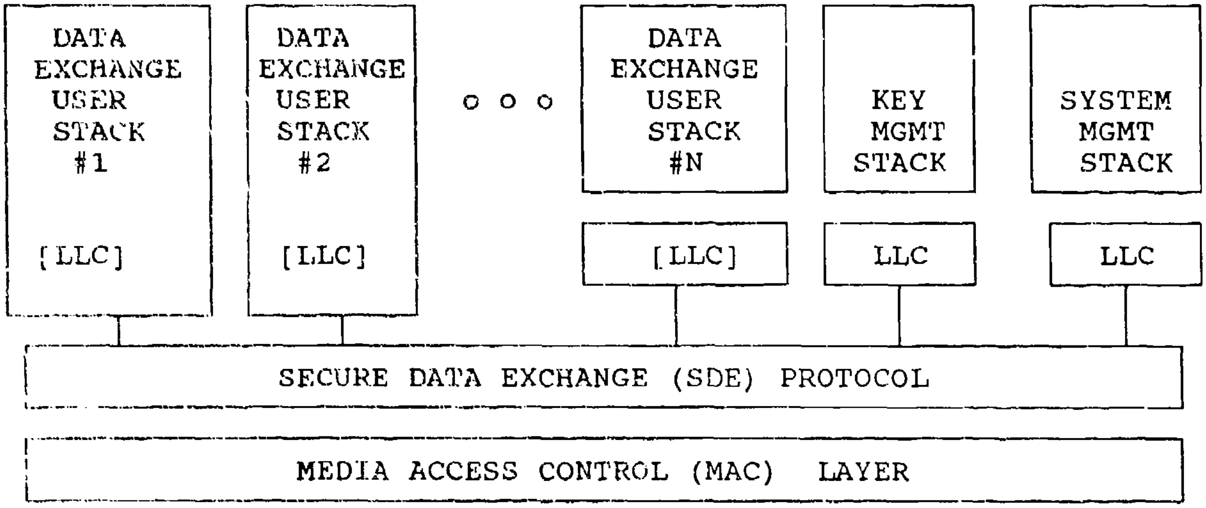

DATA EXCHANGE USER STACK #1
[LLC]
DATA EXCHANGE USER STACK #2
[LLC]
o o o
DATA EXCHANGE USER STACK #N
[LLC]
KEY MGMT STACK
LLC
SYSTEM MGMT STACK
LLC
SECURE DATA EXCHANGE (SDE) PROTOCOL
MEDIA ACCESS CONTROL (MAC) LAYER

Figure 4 -- SILS Stacks

In the "transparent" mode of the SDE protocol, this User Stack knows nothing about SILS. The Data Exchange User Process would not communicate to either the System/Security Management or the Key Management Stacks. None of the protocols in the User Stack would need to be changed as long as they maintained the normal MAC interface. A SILS device could provide a set of security services for which it was configured without the User Stack becoming involved.

3.2 System/Security Management and Key Management Stacks

System/Security Management and Key Management reside at Layer 7. The primary motivation for the placement at Layer 7 is that they can be used by protocols other than the SDE protocol. This allows other protocols, perhaps even System/Security Management, to utilize the keys and attributes provided by the Key Management Protocol. The stacks for Key Management and System/Security Management consist of the Layer 7 protocols and those protocols at other layers that are required to support these Layer 7 protocols. As work progresses in the definition of System/Security and Key Management, the particular profile will be defined.

4.0 Detailed Model

4.1 Architecture

Figure 5 depicts the overall model of the protocols and services defined by this standard. The Layer Managers are shown on the right in the figure. The management entities are the Management Information Base (MIB), the Layer Managers (LMs), the Security Management Information Base (SMIB), and the System/Security Management. Each of these will be discussed in more detail later.

Each of the protocols that will be defined by the standard are denoted by shaded boxes. These shaded boxes contain the Key Management Protocol, the Secure Data Exchange Protocol, their respective LMs, and a Mapper. These protocols will be complete standards that include conformance testing and at least one mode of operation that allows interoperability.

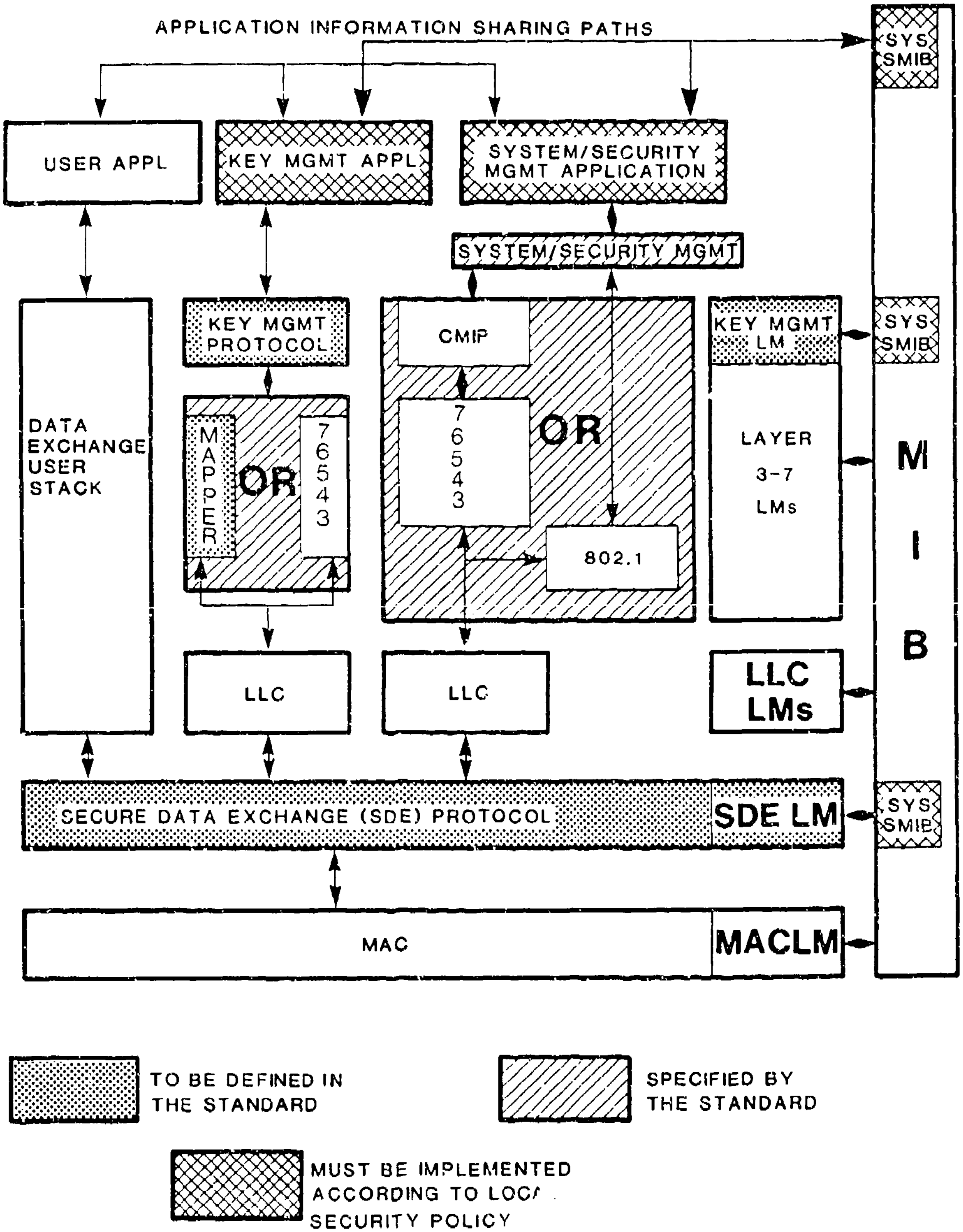

APPLICATION INFORMATION SHARING PATHS
SYS SMIB
USER APPL
KEY MGMT APPL
SYSTEM/SECURITY MGMT APPLICATION
SYSTEM/SECURITY MGMT
DATA EXCHANGE USER STACK
KEY MGMT PROTOCOL
MAPPER
OR
7 6 5 4 3
CMIP
7 6 5 4 3
OR
802.1
KEY MGMT LM
LAYER 3-7 LMs
SYS SMIB
M I B
LLC
LLC
LLC LMs
SECURE DATA EXCHANGE (SDE) PROTOCOL
SDE LM
SYS SMIB
MAC
MACLM
TO BE DEFINED IN THE STANDARD
SPECIFIED BY THE STANDARD
MUST BE IMPLEMENTED ACCORDING TO LOCAL SECURITY POLICY

Figure 5 -- Complete SILS Model

The single-hashed boxes (single lines running from lower left to upper right) indicate recommendations that will be made by the standard, but provide more than a single option for the implementor. There are three of these boxes: the System/Security Management Stack, the Key Management Stack, and the System/Security Management Box. The Security Management box represents a yet undefined protocol. It is intended to provide any security features needed by the System/Security Management Application that are not provided by CMIP or 802.1.

There are five cross-hashed boxes that represent parts of the architecture that must be implemented according to local policy: the Key Management Application, the System/Security Management Application, and three boxes for the System Security Management Information Base (SMIB). The Key Management Application and the System/Security Management Applications request services of their respective protocols, but have more knowledge about the local policy. A key change can illustrate the distinction. The Key Management Protocol can perform the key change, but only the Key Management Application would have the knowledge of local policy to know when the key must be changed.

It is also important to notice the arrows drawn between the application processes for the User, Key Management, and System/Security Management Applications. These arrows indicate that it is possible for these Application Processes to communicate although that communication is outside the scope of this standard. Generally, if the SILS protocols were implemented in an in-line device, little or no exchange between the Application Processes would be expected. In an integrated implementation; however, it might be useful to utilize this communication.

SILS must support both the ISO and the IEEE 802.1 management framework. IEEE 802.1 is defined as running directly above LLC while ISO defines management as a Layer 7 function. Although there are two different management protocols, it is unnecessary to specify two different key management protocols. SILS will specify a "mapper" that will allow a single Key Management Protocol for both environments. (Figure 5 represents this conflict in a box to the right of the Data Exchange User Stack.) If the developed key management protocol requires services provided by the upper layers of the ISO stack, but not provided by LLC, then the mapper must provide these functions. The OSI protocols and the Mapper will be specified by this standard so that the Key Management and System/Security Management Protocols can interoperate with other SILS devices.

The model assumes a MAC interface at Layer 1 and an LLC interface at Layer 2, but the protocols for Layers 3 through 7 are currently undefined for either the Key Management or System/Security/Security Management.

4.2 Key Management

The key management application (Figure 5) makes use of the services provided by the Key Management Protocol defined by the SILS. Due to export restrictions and the variety of user needs, IEEE 802.10 will make the protocols independent of the encryption and key management algorithms. As such, there must be some mechanism put into place to allow the easy identification of algorithms. Fortunately, ISO has defined a standard for a registry of encryption algorithms in DIS 9979. They are currently looking for a sponsor to maintain the registry, and there are presently three volunteers. It is very likely that in the near future, the registry service will be provided. (It is the hope of IEEE 802.10 that a similar service will be provided for key management.)

There are a number of options that must be negotiated in regard to the key that is distributed. These options include, but are not limited to:

o Key Management algorithm
o Encryption algorithm
o Access Control
o Security Services for which the key is used

Keys may provide protection for user data or for other keys. Also, keys may be used for symmetric or asymmetric algorithms. In the case of an asymmetric algorithm, a key pair is required--encrypt and decrypt.

The following protocol functions are currently defined:

1. SET -- Distribute a key and/or security attributes to another party.
2. GET -- Retrieve a key and/or security attributes from another party.
3. UPDATE -- Modify the key and/or security attributes of another party. When applied to the key, it means that a new key is generated as a function of the existing key.
4. CREATE -- Create a key relationship where there is no master key encrypting key.
5. DELETE -- Remove the key and/or security attributes of another party.

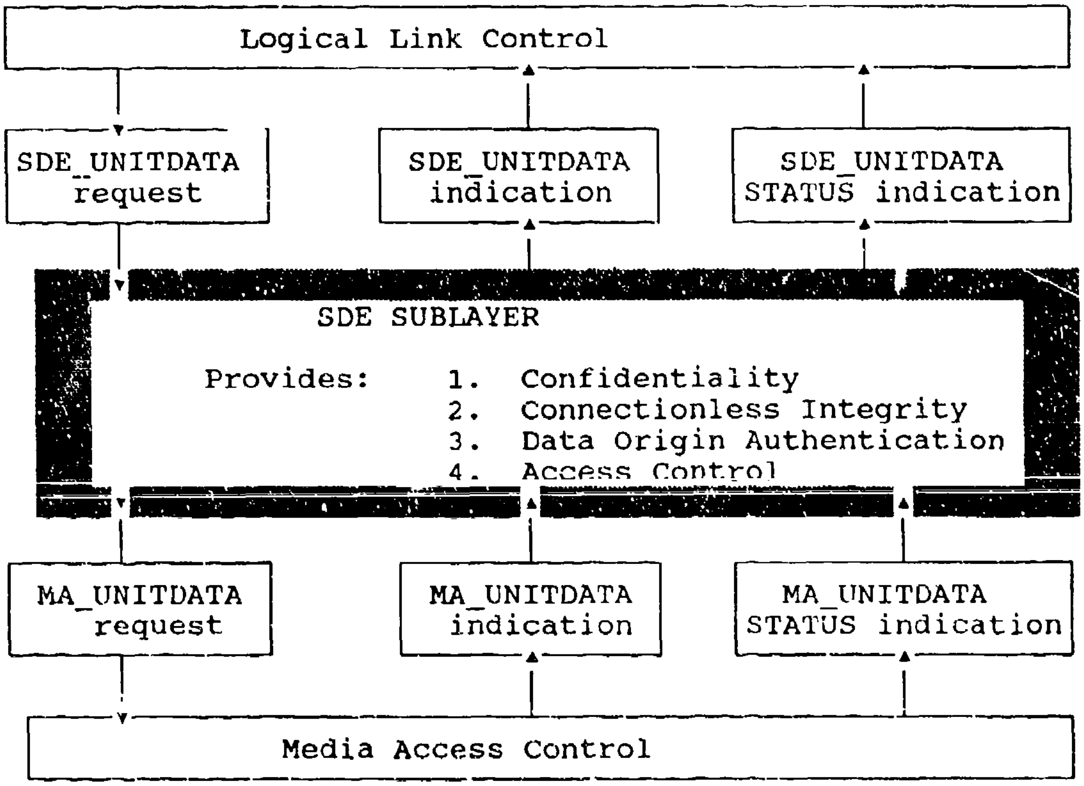


Figure 6 -- SDE Security Services

4.3 Secure Data Exchange (SDE)

The SDE protocol is placed transparently directly above the MAC layer. The SDE security services are provided using the primitives shown in Figure 6. The following is a list of the services with a brief discussion of the mechanisms used to provide the services.

o Data Confidentiality -- The SDE sublayer provides data confidentiality by performing encryption over the LLC Protocol Data Unit (PDU). The sublayer provides for the use of multiple encryption algorithms and depends on an external key management service for establishing a Data Encryption Key (DEK) and for choosing an encryption algorithm.

o Connectionless Integrity -- The SDE sublayer provides connectionless integrity by calculating an Integrity Check Value (ICV) and appending it to the end of the SDE PDU. The sublayer depends on an external key management service to establish an integrity key and for choosing an integrity algorithm.

o Data Origin Authentication -- Data Origin Authentication is achieved by the use of pairwise keys and/or by placing the Source Address in the protected portion of the security header. The Source Address also prevents reflection of the PDU.

o Access Control -- The SDE sublayer interacting with the SMIB, provides access control enforcement. For each PDU the source and destination MAC addresses are used as an index into the SMIB. If no association exists, then the PDU is discarded and the Layer Manager is notified. A similar check occurs on receipt. A set of security labels may accompany the association. A PDU that is labeled outside the security label set of the association will not be delivered and the Layer Manager will be notified.

The threats that these services protect against:

- o Unauthorized Disclosure
- o Masquerading
- o Data Modification
- o Unauthorized Resource Use.

The security services do not protect against:

- o Traffic Analysis
- o Covert Channels
- o Flooding
- o PDU Damage
- o Replay
- o Physical/Electrical Damage
- o Misordering Data
- o Undetectable PDU Header Modification.

4.4 System/Security Management

Each of the individual protocols (e.g., Secure Data Exchange, Key Management) must identify objects that they need to be managed. The Layer Manager definitions in the key management and SDE sections provide the encoding of the objects and their effects on the protocol state machines. If it is necessary for system/security management to communicate with another end-system, the System/Security Management Application Entity does so using either CMIP or 802.1. Between the protocols and the application, there are additional services that are standardized by 802.10. These services are enhancements to CMIS/CMIP and 802.1 that provide the sufficient security and functionality for the management of security to the System/Security Management Application. These services may or may not require a protocol separate from 802.1 and CMIP, so where there is a "Key Management Protocol" and a "Secure Data Exchange Protocol" in Figure 5, the protocol for System/Security Management is labelled "Security/Mgmt".

The SMIB is part of the management "cloud" mentioned in Section 3. The SDE management objects make up part of the SMIB for the end-system. Key Management and System/Security Management may modify objects in the SMIB. The effect of these modifications to each of the security objects identified by the SDE protocol is specified in the SILS.

The SMIB provides an internal communication path between the Layer 7 System/Security Management and Key Management Application Processes to the Layer 2 SDE protocol within an end-system. The modification of the SDE objects may affect the User Stacks or Key Management and System/Security Management. For instance, Figure 7 shows the Key Management Application providing key material for the SDE protocol via the SMIB. This key material may be used to protect the user information if the appropriate attributes are set.

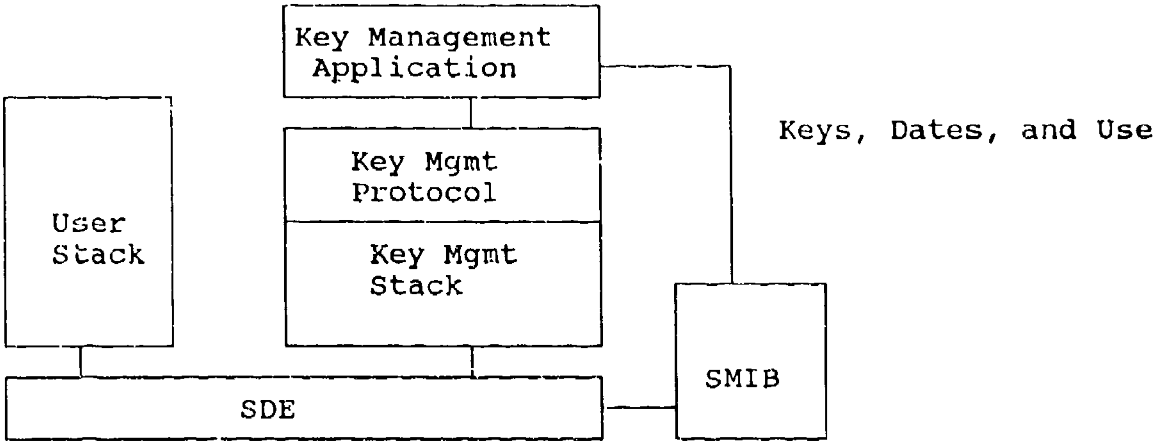


Figure 7 -- Use of the SMIB

5.0 Conclusion

At the time this paper was written, the SDE protocol was written and work on the other protocols was just beginning. It is envisioned that by the fall of 1989, the SDE protocol will be balloted as the standard, and the System/Security Management and Key Management standards should follow within the next year. The IEEE 802.10 working group is open to the public and participation is encouraged and welcomed.

6.0 Acknowledgements

This paper reflects the collective work of the IEEE 802.10 committee as interpreted by the authors. As such, the authors would like to express their appreciation to each participant. Special thanks are due Ken Alonge who helped develop the overall model graphics contained in this document.

A DYNAMIC NETWORK LABELING SCHEME FOR A MLS LAN

Peter Loscocco
Office of Research and Development
National Computer Security Center

INTRODUCTION

In designing a multilevel secure network, certain design decisions must be made that have far-reaching effects on the network's final operation. One such decision is what subset of the total possible levels of security will the network support and how will its elements be represented. Once this decision is made and the network made operational, can this subset or the representations of it be changed? Should it be changed? If so, what are the permissible changes and how will they effect the secure operation of the network? And at what cost does this extra flexibility come?

This paper will examine the need for a security level maintenance facility on a MLS LAN as well as its implications. It will show how such a facility allows great flexibility to be introduced to the security-relevant portions of the LAN, making it very sensitive to the security requirements of its users, and at the same time have minimal impact on the network operation. Part of that added flexibility is a plan for variable length security labels whose dimensions can be changed essentially "on the fly" without significantly disturbing the operation of the network. There also will be a discussion on security label translators.

SECURITY LEVELS AND THEIR REPRESENTATIONS IN THE CONTEXT OF A MLS LAN

MLS LAN Background Information

The following discussion is based on a design for a Multilevel Secure Local Area Network (MLS LAN) [1]. It is a broadband dual-cable bus LAN that uses the Transmission Control/Internet Protocol (TCP/IP) and Carrier Sense Multiple Access (CSMA) [2,3,4]. Although presented here for a specific LAN, many of the concepts to be discussed could be generalized to apply to other MLS networks as well.

The major function of the LAN is to provide a secure transport service for its attached hosts. In essence, that means that the LAN will provide a trusted communications path between pairs of hosts. The LAN will be trusted to allow only those hosts whose users have both the proper clearance and need, to establish connections. This is in accordance with the DoD security policy [5]. Furthermore, the LAN will assure that all message traffic will belong to a valid connection and only the intended host will receive the messages. Clearly, a great deal of trust would be required in the LAN, at the very least an B2-level as described in the Trusted Network Interpretati n [6], if it were ever to be used operationally. The techniques involved in providing that level of trust, however, will not be addressed in this paper.

There are no restrictions as to what kinds of hosts are permitted to use the LAN. In fact, they may range in capabilities from single-level untrusted to multilevel secure. However, if a host is to use the LAN it must be attached to the LAN via a Multilevel Secure Bus Interface Unit (MLS BIU). The BIU's will relieve the hosts from most of the burden of network processing, requiring the hosts to understand only a minimal Host-BIU protocol.

The Foundations of Security

Security on the LAN is maintained by restricting connections between hosts. When established, each valid connection has a connection type and a set of security levels assigned to it. Only data of the proper type and with a security level from that set are permitted on the connection. By restricting information flow in this way, only legitimate traffic at the proper security levels is able to enter or leave the LAN.

Since security-related decisions on the LAN are made based on connection types and security levels, it is important to understand exactly what these are. A connection type is exactly what its name implies, an identifier which denotes what type of traffic will be on a connection. As an example, two types of connections could be mail and remote login. By distinguishing the types like this, specific types of connections can be denied to a host without completely isolating it. In keeping with the example, it would be possible for a host to be allowed to exchange electronic mail with another host without also allowing remote logins. Furthermore, it would be possible to permit a host's users to make remote connections but restrict users from other hosts from connecting to it.

Security levels are the fundamental elements in capturing the concepts of classification, clearance and, to some extent, need-to-know. They may be associated with data, to denote its classification, or with hosts, to denote its user's clearances and need-to-know. Valid connections are assigned security levels based on the valid security levels for the involved hosts and the level of data intended for the connection.

Security levels consist of three pieces of information, a sensitivity value, a set of compartments, and a set of handling restrictions. A partial ordering can be imposed on the set of all security levels using the well-known dominance relation. By using the dominance relation, ranges of security levels can be referred to by using upper and lower bounds. These bounds are assigned to connections and the range implied by them determine the valid levels of data that may utilize the connection. A more complete description of security levels and their ordering can be found in [1] and [7].

There is one other way, unrelated to security levels, that access to connections on the LAN can be restricted. Access can be denied on the basis of host pairs with or without respect to connection types. This adds an extra layer of security by making it possible to prevent a host from making certain types of connections with certain other hosts regardless of classification issues.

Components of the MLS LAN

Each host is attached to the LAN via a Bus Interface Unit (BIU). It is the responsibility of the BIU's to ensure that the LAN operates securely, allowing only legitimate traffic at the proper security levels onto connections between hosts. Although responsible for the total management of connections on the LAN, the BIU's do not have the necessary knowledge or authority to approve or disapprove connection establishment. Consequently, there must be some place for the BIU's to turn to for help. That place is the Access Controller (AC).

The AC is a special host whose primary function is authorizing BIU's to establish connections amongst themselves. It bases its decisions on Mandatory and Discretionary Access Control (MAC/DAC) tables which contain all of the necessary clearance and need to know information for each BIU and BIU pair [8]. These tables reside on the AC and are maintained by the Network Security Officer (NSO). Once a connection is opened, the AC is no longer involved, and the security of the connection rests with the BIU's.

During the establishment phase of a connection, the AC instructs the involved BIU's as to what type and at what security levels the connection is to be. At this point the BIU's have enough information to perform the mandatory access control checks on all the data sent or received on that connection. It is a requirement that all data have a label that properly reflects its connection type and security level. The BIU's use these labels, along with the information provided by the AC, to do the security checking for each connection. Only data labeled with the proper connection type and a security level falling within the prescribed range of security values are permitted to be sent or delivered.

The Representation and Use of Connection Types and Security Levels

To a large extent, the secure operation of the LAN depends on connection types and security levels. Although well defined, these concepts are still quite abstract. In order to be used, they must be represented in some more concrete way that the LAN can interpret. It will be these representations, and not the abstractions, that will be used throughout the LAN to ensure security. Unfortunately, this places some limitations on the LAN.

Theoretically, there is no limit to the possible number of connection types and security levels, and in the ideal situation, a MLS LAN would be able to support all of them. In reality this is not possible. Even with modest choices for the numbers of connection types and elements of each component of the security levels, the total number can become prohibitively large when all the combinations are considered. The time and space required to efficiently process and uniquely represent them is too costly to the performance of the LAN. Consequently, only a subset of the total possible connection types and security levels can be supported by the LAN at once.

There are numerous ways in which security levels could be represented, each with its own relative merits. An optimal format would be one which required a minimum amount time to compute dominance relations and a minimum number of bits to represent, was the least complex, and could represent the maximum number of security labels. Unfortunately, these criteria are at odds. The LAN designers have to make some compromise in these areas to determine a format that keeps LAN performance at a maximum, cost at a minimum, and, perhaps most important of all, satisfies the needs of the users.

The primary occurrence of connection type and security level representations is in security labels. The security label is a physical tag associated with each packet on the LAN that identifies the security level of the data contained in the packet and the type of connection on which the packet belongs . A typical format for the Network Security Label, as it is called, might consist of four fields, one each for connection types, sensitivity values, compartments, and handling restrictions. The first two fields would contain an actual value and the later two would be bitmaps with one bit position to indicate the presence or absence of each of the respective elements (fig. 1). A similar format to this has been previously used to represent security levels on a secure system [10].

If at all possible, the Network Security Label will be contained in the Internet Protocol (IP) header, specifically, in the two IP security options [3,9]. The DoD Basic Security Option, option 130, will contain the representation for the sensitivity portion of the security level and the the DoD Extended Security Option, option 133, would contain the representations of the connection types, compartments, and handling restrictions.

Con. Type	Sensitivity	Compartments	Hand. Restrict.
value	value	bit-map	bit-map

FIGURE 1: FIXED LENGTH SECURITY LABEL

Since the Additional Security Information field of option 133 is not yet fully defined for all the Protection Authorities, the actual contents of option 133 cannot be stated with absolute surety. If this option does not provide fields for all the necessary information, then provisions would have to be made elsewhere in the protocol suite.

Network security labels are not the only security labels used in conjunction with the LAN. All multilevel hosts, and perhaps even some single level but trusted hosts, will have internal security labels for their data. These Internal Host Labels, like the network labels are for packets, are the physical representations of the security levels of data on that host.

The set of security levels supported by any given host, in general, is going to be a much smaller subset of the set of total possible security levels than the set supported by the network, but, in general, a subset nonetheless. Different hosts most probably support different subsets. For instance, some hosts may support compartments and handling restrictions; others may not. Differences like these are unimportant as long as the security levels are drawn from the same total set.

Even though it is true that designers of secure computer systems recognize the need for security labels for data internal to their systems, it unfortunately is not true that they have agreed on a standard method for constructing those labels [10]. In fact, it is probably not to far from fact to say that there are and always will be as many different forms of security labels as there are vendors building secure systems. As unfortunate as this may be, its a fact of life that network designers are going to have to live with.

Since there are so many different ways that security levels will be represented on hosts and only one way that they may be represented on the LAN, there must be a mechanism that will map the hosts' representations uniquely into the LAN's and back. Without such a mechanism, the classification level of data could not be preserved as data is transferred from one system to another across the network. Obviously this label translator has to be familiar with both the host's and the network's representations of the supported security levels. This situations is analogous to the the Network Virtual Terminal concept in which all hosts on a network use a standard network character set regardless of any internal character representations [11].

There are two choices as to where the label translators should be located, on the hosts or on the BIU's. A case can be made for either choice, but the stronger of the two is for label translators residing in the BIU's. In that way, the burden of network processing is further removed from the host, as it should be. More insight into the label translation function will be given later.

A portion of the LAN that vitally depends on the representations of connection types and security levels is the security checking mechanism inside the BIU's. In order to perform its function, this mechanism must know the size and location of the representations in the data stream. Also the BIU's, as part of the state information kept for each connection, store the representations of the connection type and the upper and lower bounds of the valid security levels for both the incoming and outgoing traffic. It uses these to determine if data may flow on that particular connection. The representations do not theoretically need to be the same as those for the network security labels, but to minimize the time required for security checking should be.

After a BIU receives a message from its attached host and checks the security information against the type of connection and range of valid security levels for the intended connection, the BIU can format the security information into the network security label that is to be attached to the outgoing packets. Packets arriving from the network are handled in a similar way. The security information is extracted from the network security label and checked against the stored values. Only those packets of the proper type and with a security level falling within the acceptable range will be forwarded to the host.

As described, the BIU security-checking mechanism will work properly if the attached host has security labels that can be trusted. But what if the host does not support security labels or, equivalently, supports security labels that cannot be trusted? Fortunately the LAN will still be able to operate in these situations. As a matter of policy, such hosts are assumed to be operating in a system-high mode. The BIU's will know at what security level all connections would, by definition, have to be. In this mode, the network security label for all data leaving the host automatically will reflect the system-high level of the host.

The AC validates a connection's type and security level ranges during its establishment phase. BIU's forward connection open requests to the AC to determine if that connection is in accordance with the network security policy. The AC must check the open request against the appropriate MAC tables. Only then may the AC authorize the BIU's to open the connection. As might be expected, the MAC tables are constructed from connection type and security level representations.

A MAC table is kept for each host that may use the LAN. These tables store only the clearance portion of the information that the AC needs in order to approve connections. It is from the DAC tables that the AC determines with which hosts a given host may communicate. Conceptually, these tables can be visualized as a complete list of the security levels at which a host may communicate for each type of connection. Actually, these connections are not enumerated, and the table contains only as many records as are needed to encompass all the valid connections.

Each record in a MAC table consists of three fields, the first for the type of connection, and the remaining two for the ranges of permissible outgoing and incoming security levels for that connection. Each of these ranges is a pair of security levels, a minimum and a maximum, where the first is necessarily dominated by the second. If a host is multilevel secure, then its MAC table can have essentially any number of records. Single-level machines, on the other hand, can, at most, have the number of available connection types. The security level ranges on these would have to be set so that all outgoing traffic would be at the system high level and all incoming would be at most the system-high level.

When a connection open request arrives at the AC, its security checking mechanism extracts from the request the pertinent information needed to make a decision. The two involved MAC tables are checked for the existence of a record which could authorize the connection. If one is found in each of the two tables then the connection would be in accordance with the mandatory access control portion of the security policy. The connection may still be denied, however, based on discretionary access or resource constraints.

Like the BIU's security checking mechanism, the AC's must know the size and location of the information it needs to access. It also must know the representations of the information in both the MAC tables and in the open requests. Although theoretically unnecessary, it would, however, seem most efficient to make these representations identical.

As presented, the security of the LAN rests heavily with the representations of security levels and, to a lesser extent, connection types. Without them it would be impossible to enforce the security policy of the LAN. A very important question still needs to be answered however. Where do these representations originate?

The Static Nature of the LAN

In the past, the conventional approach to incorporate security level representations into secure systems has been to define them and build them into the fabric of the system [10]. Designers decided exactly how many security levels their system could support, making a compromise between the expected needs of the potential users and practical concerns such as space constraints and system performance. They then defined the representation for each level, not necessarily attributing it to any specific security level, and effectively hardwired them into the system. At that point the initial design decisions, good or bad, are set in stone.

Applying this approach to the LAN would mean anticipating the smallest number of connection types and elements of each security level component that could satisfy the requirements of all the potential users of the LAN. The builders would then define the formats, traditionally a fixed size, as well as the representations for the network security labels and the MAC/DAC tables. The assignment of actual security levels to the representations, ensuring that the dominance relation in maintained, would occur at a later time.

The implementation of the security checking mechanisms for the AC and BIU's, whether in hardware or software, is completely dependent on the previously defined formats and can be straightforwardly built into the system. As for security label translations, the hosts could be forced to adhere to a rigid front-end protocol, meaning label translation would be done on the host. In any event, they too depend on the formats and, with respect to connection type and security level representations, the design would be static and not easily changed.

WEAKNESSES IN THE CONVENTIONAL APPROACH

An Inability to Meet the Changing Needs of Users

The most important, yet most difficult, part of the entire process would seem to be anticipating the connection type and security level needs of the LAN's end users. If the LAN is to provide quality multilevel service to its users, then it is imperative that it be capable of maintaining the types of connections that they require at the appropriate security levels. Furthermore, the performance degradation introduced by the security features should be minimal.

Using the conventional static approach to the representations would probably work fine, provided that all the users needs were properly forecasted. There would be no problem with the service types at the proper security levels. As for performance, it would not be optimal, due to the potential for wasted bandwidth brought on by the fixed label length, but probably acceptable. But what are the chances of the LAN designers' predictions being correct?

For a single system whose security levels consist only of a sensitivity value, making the correct decisions is complicated enough, but compounding the problem by adding an arbitrary number of other systems and adding compartments and handling restrictions to security levels makes choosing the correct security level subset next to impossible. Even on the remote chance that the guess were on the mark, the needs of users are varied and subject to change.

There are essentially two kinds of changes that may need to be made to the subsets of the security level and connection type sets used by the LAN: additions and deletions. For instance, additions may be needed if some users of a currently attached host require new compartments or hosts with requirements for previously unneeded handling restrictions are added to the LAN. On the other hand, certain elements may become obsolete and have to be deleted when hosts are detached from the LAN.

As a further example, consider the sensitivity values provided for in the IP security option 130. At present there are only the four U.S. values defined. In the future, it is conceivable that other, perhaps foreign, values will be needed. There also are currently only four protection authorities recognized in IP Security Options 130 and 133. As others are incorporated, changes could be required.

It is not unreasonable to expect changes such as these to be necessary throughout the lifetime of the LAN. Even though they might be infrequent, a MLS LAN should be flexible enough to accommodate them. This would not be the case for a LAN designed using the approach described above. In fact, it is a fundamental weakness of such a LAN that these types of changes are difficult to make.

It is the LAN's static nature that makes this so. Since those portions of the LAN that depend on the representations of security levels and connection types would essentially have been built into both the hardware and software, any changes to the security level or connection type subsets would have to wait for the next version of the LAN. The designs for all the affected components would have to be modified so that the changes could be incorporated into the LAN. This could require a great deal of time and effort, not to mention money, and definitely necessitate recertification. This is not only infeasible but unacceptable, especially if frequent changes occur.

Without eliminating the static nature of the LAN, it is still possible to anticipate this problem and make allowances for it. Extra representations can be defined, but not assigned, only to be used if additions to the subset are ever needed. The representations for elements no longer needed would still exist but never be used. Using this approach the LAN would appear to be able to accommodate its user's needs for change.

Although possibly adequate, this approach is certainly not optimal. At best, it is a temporary solution. When the extra representations are eventually assigned, the original problem returns. Also, those that are deleted never can be reassigned to other values, thereby wasting part of an already limited number of representations.

Problems with Performance

A LAN with static connection type and security level representations has other problems than its inability to conform to the changing requirements of its users. There are inherent inefficiencies, and perhaps the biggest example of this is in the network security labels. Consider the example security level representation given earlier. If there were a large number of compartments, but only very seldom were more than one compartment associated with a packet at once, there would be a large amount of wasted bandwidth when transmitting the security labels. The bitmap for the compartment field would have to be large enough for all of the compartments, and only a few bits at a time would ever be set. The problem is worse if, on that same network, two hosts communicated but had no need for compartments. In this case, the entire compartment field would still have to be tacked on to all the packets just as useless baggage.

What About Label Translators?

It is the purpose of a label translator to uniquely transform security levels represented in a host's internal format into the network format and back again. At first glance, this may seem like a trivial issue but, upon closer examination, proves quite the opposite. What complicates the label translators job is the lack of a labeling standard. There are different labeling formats for different machines. It is possible to have a situation where different representations for the same security levels, or even worse, the same representations for different security levels exist.

How is the function of label translation affected by the conventional approach described above? If it were done in the BIU's, it would seem that the label translators would be subject to the same problems as the security label representations as far as the difficulty involved in incorporating changes. Also there would have to be different label translators for different machines and development of new translators would have to be done for any new machines. There is also another problem. Most likely the translator would be implemented as some sort of table look up. This table, unless it were able to be modified, would limit any changes that could be made to a host's MAC tables.

What if the translations are done on the host? This has several disadvantages too. First, this task is an extra burden on the host. One reason for having BIU's in the first place is to remove such burdens. No longer can a host just pass its own security label along with the data. The second disadvantage is a security concern and necessitates returning the label translators to the BIU's where they belong. In providing specifications for label translations in the host to BIU protocol, too much information could be inferred about the internal mechanisms of the

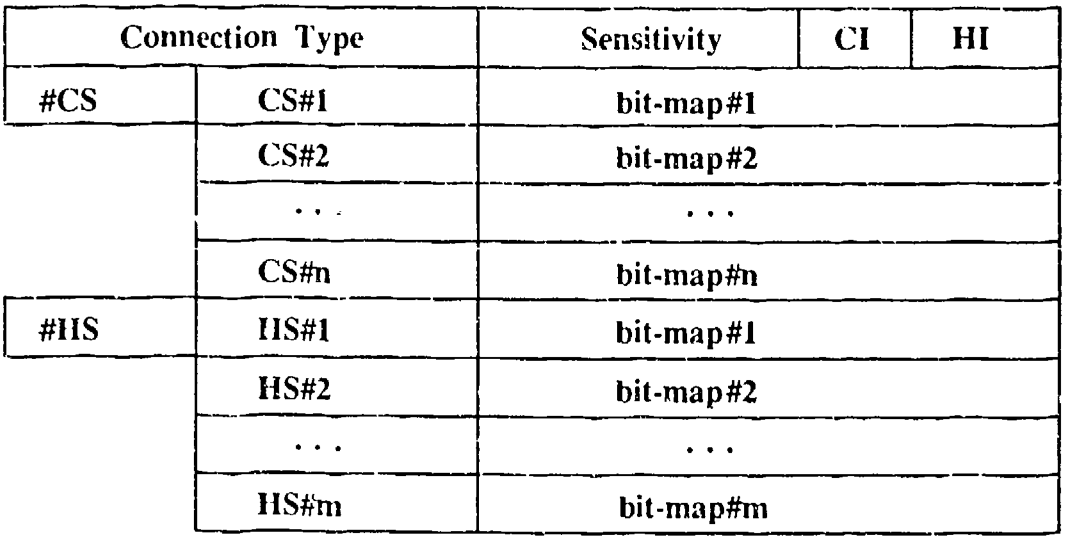
BIU's. A greater concern, however, is in the sensitivity of the actual security levels themselves. Many consumers of security levels will not tolerate a disclosure of even the names of these levels.

A DYNAMIC NETWORK LABELING FACILITY

The Introduction of a Variable Length Network Security Label

Thus far it has been shown that the static properties of the LAN can lead to a workable but certainly less than optimal network in terms of functionality, usability, efficiency, and to some extent security. It is possible to improve the situation by replacing the fixed length network security label with a variable one. One possible format is a simple variation of the earlier example (fig. 2).

There would still be the four components to the security labels, and the connection type and sensitivity fields would be the same. The difference comes in the way compartments and handling restrictions are handled. The entire set of compartments is broken up into equally sized subsets, each being assigned a unique indicator. Each element of these subsets is assigned a position in a bit-map, one bit-map per subset. The compartment component of the security label is then just pairs of subset bit-maps and indicators preceded by a field to identify the number of pairs. There is also one additional bit position which would be set only if compartment information were included in the label. The handling restriction field would be treated in an identical manner. It is important to note that t' lengths of the various fields vary with the number of elements in each of the subsets and that in the degenerate case, where there is only one subset, this format is essentially the one originally discussed.



Connection Type		Sensitivity	CI	HI
#CS	CS#1	bit-map#1		
	CS#2	bit-map#2		
	• • .	• • •		
	CS#n	bit-map#n		
#HS	HS#1	bit-map#1		
	HS#2	bit-map#2		
	• • •	• • •		
	HS#m	bit-map#m		

FIGURE 2: VARIABLE LENGTH SECURITY LABEL
CI and HI = Single bits set when compartments and handling restrictions are present in label;
#CS = The number of compartment subsets; CS#i = Subset indicator for bit-map # i;
#HS = The number of handling restriction subsets; HS#i = Subset indicator for bit-map # i

The greatest advantage to such a scheme is that many more security levels can now be represented in a smaller amount of space. It is now conceivable that hundreds, or thousands, of compartments and handling restrictions can be supported on the LAN at once where before such numbers were prohibitively large. If the number and size of the subsets are in·ᵣlligently chosen, then most if not all of the security levels actually used on the network could be represented using at most one subset. This minimizes the size of the label and reduces the wasted bandwidth significantly. This scheme also is not incompatible with the IP Security Options as they are currently defined.

There is really only one disadvantage to using this variable length label. It is more complex. The savings in reduced transmission time the shorter label brings could be offset during dominance checking and label translations But if recent trends in increased processing power continue, that lost time will be regained. It also should be mentioned that this design actually adds inefficiency to the security label if only small numbers of compartments and handling restrictions are needed.

A Security Level Maintenance Facility

Although the introduction of variable length security labels to the network addresses some of the performance concerns stated earlier, it does not solve the problem brought on by the changing security needs of network users. This section will show that by providing a security level maintenance facility along with the variable security label, this problem can be addressed and performance actually further increased.

A security level maintenance facility is a means by which the LAN learns about security levels and their representations. No longer is this information built into the hardware and software. This facility is to reside on the AC and be used by the NSO. When configuring the LAN, the NSO enters the needed information, and all aspects of the LAN that depend on this information are automatically initialized. As changes become necessary, the NSO can make them, and, as before, all affected portions of the LAN are automatically notified and updated.

What this means is that when the security level needs of the network users change, they can be accommodated promptly, inexpensively, and at little or no interruption of service to the LAN. There should be no need for recertification as long as all changes are made in conjunction with prescribed policy.

Such a facility gives another aspect of flexibility because changes also could be made to the makeup of security levels. For example, if, for one reason or another, it were decided not to do access control based on connection types, that field could be eliminated. The only field that is absolutely necessary is the sensitivity field.

It also is possible to improve performance. If, during the monitoring of the network, it were noticed that frequently more than one subset was required to represent compartment information, then the makeup of the subsets or their size could be adjusted until that frequency was reduced. In the same way, if multiple subsets never occurred, then the sizes might be reduced making more subsets and shorter labels. This would in a sense be a mechanism to fine tune the LAN by reducing the average security label size.

For this maintenance facility to function, some changes have to be made to the AC and BIU's. The AC will have to keep other databases besides the MAC and DAC tables and support the interface for the NSO. The security checking mechanisms on both the AC and BIU's will have to be made more flexible, and the BIU resident label translators now become dependent on the AC.

A parameter database will have to be maintained on the AC. This database will keep information such as the maximum numbers of each of sensitivities, compartments, handling restrictions, and connection types, (and if each is supported). Also the dimensions of the subsets and subset indicators, as well as, lengths of other fields needed in the security labels and checking mechanisms. This data will be referred to whenever changes are made to ensure consistency.

There will have to be a database for the security levels and their representations. The elements of each of the components of security levels need to be kept here along with their representations. In the case of compartments and handling restrictions this is a subset indicator along with a bit position. There is also a database for the connection types for similar information. Both of these databases depend on the parameter database and cannot be constructed without it.

There are two databases associated with connected hosts. The first, the host-type database, contains information on different types of hosts and not specific machines. Among other information, the machines' internal security label representations, if any, are kept here. This information does not include the assignments of security levels to those representations. The second database, the host database, is for pertinent information on all hosts on the LAN. This database depends on the both the host-type database and security levels database and includes such information as the host type, security level assignments to the representations, if any, and the trustability of labels coming from that host. Label translators will depend of this database.

All of these databases have to be present and consistent before the LAN can be made operational. The MAC and DAC tables depend on all of them both for format and the entries. The AC's security checking mechanisms must be flexible enough to handle changes in the formats of the MAC and DAC tables. The mechanism remains fundamentally unchanged but becomes a function of the security level parameters. When the AC is started, its checking mechanisms must be initialized from the parameter database. The same is true for the security label checking mechanism and label translators in the BIU's. When a BIU is started, it needs to know the level of trust it can place in its attached host, the format of the network security labels, and how to translate between them and the host's. For all this, the BIU depends on the databases on the AC.

The NSO is responsible for managing all of the AC databases and needs to have a trusted means for doing so. The security label maintenance facility provides this to the NSO. It ensures that when the databases are constructed, all the dependencies are adhered to. Furthermore, it will force the NSO to verify that, for all the supported security levels, if one security level dominated another, then their representations will preserve that relationship.

Before the LAN is made operational, it is fairly straightforward to keep everything consistent because everything is resident on the AC. Changes can be made without any impact. This, however, is not the case with an operational LAN, and every change to any database could have great impact.

Changes could affect the MAC and DAC tables, security labels and the label translators and thus ongoing connections. For instance, if the LAN were currently supporting the maximum number of security levels for the current label size and it were deemed necessary to add others, the parameter database would have to be changed to allow for the expansion. This has the side effect of changing the label size and all of the security checking mechanisms. Also, if new security levels were added or even if the MAC table for a host were changed, this would necessitate updating the BIU's label translator. If security levels were deleted on the AC, it is possible for a legitimate connection to suddenly become invalid.

Whenever the NSO attempts to alter any of the maintenance facility's databases, the facility must ensure that the proposed changes preserve the consistency of the databases before the changes can be made. To do this, it must examine all the database entries, using the known database dependencies, that possibly could be effected by the change for potential illegal modifications. When conflicts are detected, the facility allows the NSO to ignore the original request or attempt to resolve the conflicts interactively.

As an example, consider the case where the NSO wants to alter a host's MAC table to allow it to communicate using additional compartments. Before this could be done, the maintenance facility must determine if the compartment even exists and if there is a record of that host's internal representation for that compartment that can be used in label translations. If either of these conditions is not met the NSO is given a chance to rectify the situation by allowing him to enter the information into the appropriate databases. But these changes also must be checked. What if, by adding the new compartment, the maximum number of compartments would be exceeded? Once again the NSO will be given the chance to correct the inconsistency by increasing the maximum parameters or cancel all the current proposed changes.

As a second example, consider a situation where the NSO is trying to delete a compartment. This cannot be done until all references to that compartment in the dependent databases have been eliminated. These include entries in the MAC tables as well as entries in the hosts database. As before the NSO can deal with the potential inconsistencies as they are detected. If all the dependencies in the databases are considered when alterations are made, then they should be able to be kept consistent.

The maintenance facility's job is not completed as soon as it determines that the AC databases can remain consistent after the NSO's proposed changes are made. It must examine the state of the network for potential problems brought on by the changes. If any are detected, the facility must reach out onto the LAN to correct them. Consider the previous two examples.

In the first, an addition was made to a host's MAC table. Before that host could possibly open a connection using the new compartment, its BIU would have to receive an update to its label translator, a relatively minor effect to the LAN. But if, as suggested, the maximum number of compartments had to be increased, the effects to the LAN are more significant By increasing the maximum compartments, the number and/or size of the compartment subsets would change. This could, in turn, alter the size of the network security label. If so, the facility would have to adjust the parameters for the security checking mechanisms on the AC and each of the BIU's along with their label translators.

In the second example, deleteing a compartment would necessitate the updating of the label translators of all the BIU's that previously could use that compartment to reflect the change. What if there were active connections at that level? As part of the NSO's total capability, there would be a facility to terminate connections. The maintenance facility would have to evoke that power to terminate what would now be an illegal connection. As a more considerate alternative, the facility, at the disgression of the NSO, might be allowed to wait for the connection to terminate naturally.

A Belittlement of Potential Drawbacks

When the security label maintenance facility is modifying portions of the LAN, service may be temporarily suspended. The magnitude of the changes will determine the length and extent of the interruption. If for instance the change only affected one BIU, only that BIU's traffic would be delayed. On the other hand, if the makeup of security labels were to be changed, then all traffic on the LAN would be stopped until the change was completed. Actually traffic between some hosts would be able to continue until one of the involved BIU's is notified of the changes At that point communication would halt and not resume until both had been notified.

The biggest drawback to being able to dynamically update those aspects of the LAN that depend on security labels would perhaps be this interruption in network service. If the network were large with lots of open connections, some changes could take a while and delays would be long. Requests for new connections would have to wait adding further to the delay. There is also the question of the extra burden on the AC. Can an already busy AC handle the extra work of supporting the maintenance facility?

Fortunately, such changes should be infrequent and, when made, are in support of legitimate user requirements. Most changes would fall into the category that do not have wide-spread effects on the LAN. Changes, such as in the earlier example of increasing the maximum number of compartments, that have great impact might never be needed, especially if the initial system parameters were on the mark. The facility, however, would be there if needed.

The delays and extra burden to the AC should be infrequent too by the same reasoning. Attempts also can be made to minimize interrupted service. For instance the AC could analyze the current state of the LAN and notify the BIU's in an order that would keep their individual waiting times to a minimum. Or, if it were not imperative to effect the changes immediately, they could be queued until LAN usage was low, perhaps late at night.

The NSO could also help minimize the service interruptions by intelligently choosing the order of his actions. If he were to make changes as a series of smaller changes rather than one large one, then the computation involved could be greatly reduced. For example, if instead of deleting a compartent that many hosts still potentially might use and letting the facility clean up the databases, the NSO should delete it from each host individually before attempting to edit the security level database. This probably is closer to what might actually happen. There probably would be no pressing need to delete that compartment unless all the hosts on the network no longer used it.

There are other drawbacks as well. The implementation of the security level maintenance facility would not be an easy task. It adds to the complexity of the AC and the BIU's. This will raise the end cost of the LAN and complicate any certification efforts. What policy guidelines would have to be in place to ensure that all changes by the NSO are appropriate? Is the added utility and flexibility worth the added expense and effort? The answers would have to rest with the potential users of the MLS LAN.

CONCLUSION

In this paper it has been shown that having a facility for maintaining security levels on a MLS network is not only desirable but quite feasible. Also it has been shown that when used in conjunction with a variable length security label such as the one presented, that in addition to an added ability to conform to the changing security needs of users, an increase in performance might be achieved. Although incorporating this security level maintenance facility into the AC of the MLS LAN does have an effect on complexity and performance, it should be manageable and hopefully have minimal impact on the day to day operation of the LAN. The concepts decribed here currently are being implemented, and will be tested, at the National Computer Security Center.

REFERENCES

[1] P. Loscocco, "A Security Model and Policy for a MLS LAN", *Proceedings of the 10th National Computer Security Conference*, September 1987.

[2] *Military Standard: Transmission Control Protocol*, MIL-STD-1778, August 12, 1983.

[3] *Military Standard: Internet Protocol*, MIL-STD-1777, August 12, 1983.

[4] *The Ethernet, A Local Area Network: Specifications*, Version 2.0, Digital Equipment Corp., Intel Corp., and Xerox Corp., November 1982.

[5] DoD 5200.1R, *The Department of Defense Information Security Program Regulation*, July 1982.

[6] *Trusted Network Interpretation of the Trusted Computer Security Evaluation Criteria*, NCSC-TG-005, 31 July 1987.

[7] *CDRL 145, Formal Draft Subsystem Design Analysis Report - Engineering Report: LAN Interfaces*. GTE Contract No. F19628-84-C-0052, 10 August 1982, Volume 4, Appendix C.

[8] *Department of Defense Trusted Computer Security Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[9] *RFC 1038, Drevised IP Security Option*, M. St. Johns, IETF, January 1988.

[10] *SCOMP Trusted Facilty Manual*, FSD-85-11-5, Honeywell Federal Systems Division, May 1986.

[11] *Military Standard: Telnet Protocol*, MIL-STD-1782, May 10, 1984.

Extending Mandatory Access Controls to a Networked MLS Environment

R. S. Arbo
E. M. Johnson
R. L. Sharp

AT&T Bell Laboratories
Whippany, New Jersey 07981

ABSTRACT

We present a design of a software package that allows multi-level secure (MLS) systems to securely communicate without modifying or trusting the existing network applications. The package resides in the security kernel and provides label passing, secure session setup, network trusted path, and auditing. Also included is a description of an automated interface to a STU-III encryption modem.

INTRODUCTION

Mandatory Access Control (MAC) involves three things: an access request, the label of the subject, and the label of the object. When the subject and object reside on the same host these pieces of information are readily accessible. When the subject and object are on different hosts the task becomes harder. Typically a new subject is created on the host where the object resides to represent the original subject. The label of this new subject must unambiguously[1] represent the label of the original subject at all times. If not, MAC policy may be violated. In addition, this pairing of subjects should be recorded in the object host's audit trail. This will allow a security event to be traced back to the actual user.

Most networking software does not recognize labels or concern itself with audit trails. We could rewrite the networking software to include these capabilities, but then we would be trusting this software, which would increase the size of the Trusted Computing Base (TCB). In addition, it would require a large maintenance effort as new releases of the software came out as well as delaying these releases.

Our solution is to use the capabilities already offered by a Multi-Level Secure (MLS) host to build a security wall around the networking software. The networking applications remain untrusted and unaltered. After the initial connection is made, the two kernels exchange labels and audit information. We have developed a software module called TSES (Trusted Sessions) that is implemented in the kernel to provide this functionality. The module is described in section 2.

The solution still requires the user to provide authentication information (password) to the called host.[2] This authentication information would normally pass through the untrusted network software on the caller side. For this reason a trusted path capability has been developed to bypass the network software for transmission of sensitive information. The mechanism that provides this function (TPATH) is described in section 3.

We make several assumptions about services provided by the underlying network software and hardware. These assumptions are outlined in section 2. To provide a complete solution we developed a secure point-to-point network that satisfies these assumptions. The network is based on the AT&T STU-III terminal[1]. The STU-III is an encryption telephone/modem approved for transmission of all U.S. Government classified information. The interface to the STU-III is described in section 4.

[1] Orange Book terminology.

[2] In this paper we refer to host initiating the call as the "caller" and the host receiving the call as the "called".

The secure operating system used for this project was AT&T's System V/MLS.[2]. System V/MLS is a secure version of AT&T's UNIX® System V operating system.[3] Many of the details of the implementation are UNIX-specific and some knowledge of the UNIX operating system is assumed throughout this paper. However, the concepts used and discussed in this paper should be relevant to any MLS operating system.

TSES (TRUSTED SESSIONS)

TSES is a sessions-oriented label enforcement driver. It is designed for use in full duplex network or point-to-point communications. A TSES drive is required on both the caller host and the called host. TSES's purpose is to unambiguously pass the caller's label and other identification information to the called TSES, and to restrict the server in the called host to operate only at this label. The information is passed via an exchange between the caller and called TSES.

High Level Issues

In this section we discuss the high level characteristics of TSES and some problems that are general to the design of MLS network interfaces.

Assumptions of network services - TSES trusts the network below it. It assumes the following service features:

- Error free data transmission.
- Data arrives at the destination in the order it was written.
- A "closed" network, that is, the hosts that can be contacted from any TSES port will have a TSES on their port(s).
- Data confidentiality (secrecy), protection against disclosure of data to any but the intended recipient.
- Data Integrity

Although the above requirements are certainly not trivial, they are feasible and could be satisfied by a closed TCP/IP[3]. network.

Transparency and portability - TSES is transparent to applications and networks. TSES is also highly portable. Portability requires that TSES make the fewest assumptions about the behavior of applications and networks. TSES makes no assumptions about the behavior of the network, only the services it provides. The only assumption about the behavior of the application is that the server executes a modified version of the UNIX *login* program.

Protocol Hierarchy - TSES sits on top of a transport provider. This can be a "real" transport provider like TCP or it might be a lower layer as in the case of network stacks that do not require a transport layer, for instance, an RS-232 port driver. More important than TSES being at any particular layer, TSES should be placed directly underneath the networking application.

Restrict network session to a single label - Because network connections allow data transmission in both directions, both sides of any connection must have equal labels to avoid violating the security

[3] System V/MLS entered formal NCSC evaluation for the B1 Orange Book rating in October of 1988 It is expected that final award of the B1 rating will occur in October, 1989 System V/MLS is the first portable secure operating system and has been ported to several vendor's architectures.

policy. For instance, if the caller were labeled UNCLASSIFIED and the called were labeled SECRET, reads initiated by the caller would violate "no read up" and writes initiated by the called would violate "no write down."[4].

Trusted listener - The listener process over TSES on the network port must perform several functions which require it to be trusted. These functions include: caller authentication, relabeling the network port to the label obtained by TSES, executing the requested server with the appropriate label, and restricting the caller's session to that single label. The System V/MLS version of *login* already takes care of these tasks. Only minor additions were made to support TSES. This restricted network applications to only those that used *login*.

Network labels - System V/MLS supports labels with up to 255 levels and 1024 categories. It would be impractical to require a uniform interpretation of labels across all hosts on a large network. Furthermore, even if this were required, it would be difficult to enforce. To alleviate the situation, TSES provides network labels that are uniformly interpreted across the network. Then instead of the caller TSES passing a host specific label, it passes the network representation of this label. On the called side, the network label is mapped to that host's equivalent label. If the called host does not have an equivalent label, the connection is denied.

Outgoing port - A problem arises when trying to support outgoing ports on a multi-level secure system. Since the invoker opens the port for both reading and writing, the invoker's label must match that of the port. It would be wasteful to restrict a port to a single label since a system could have up to 60,000[4] unique labels. Therefore, the port must be accessible from a variety of labels (labeled subjects). However, allowing simultaneous access to the same port by subjects of unequal labels would violate security policy since one subject might be able to read another's data. Our solution is to save the label of the subject that first opened the TSES port. If any other subjects subsequently open the port (before the first subject closes the port), TSES checks their label for equality with the first subject's label. For example, if a SECRET and an UNCLASSIFIED process both tried to open the same outgoing TSES port, the process that got there first would succeed and the other would fail.

Implementation - TSES has been implemented in two versions. It has been implemented as a STREAMS[5] driver linked over the transport provider and as a character-based pseudo driver for use over non-STREAMS drivers and ports (e.g., RS-232).

Mechanism

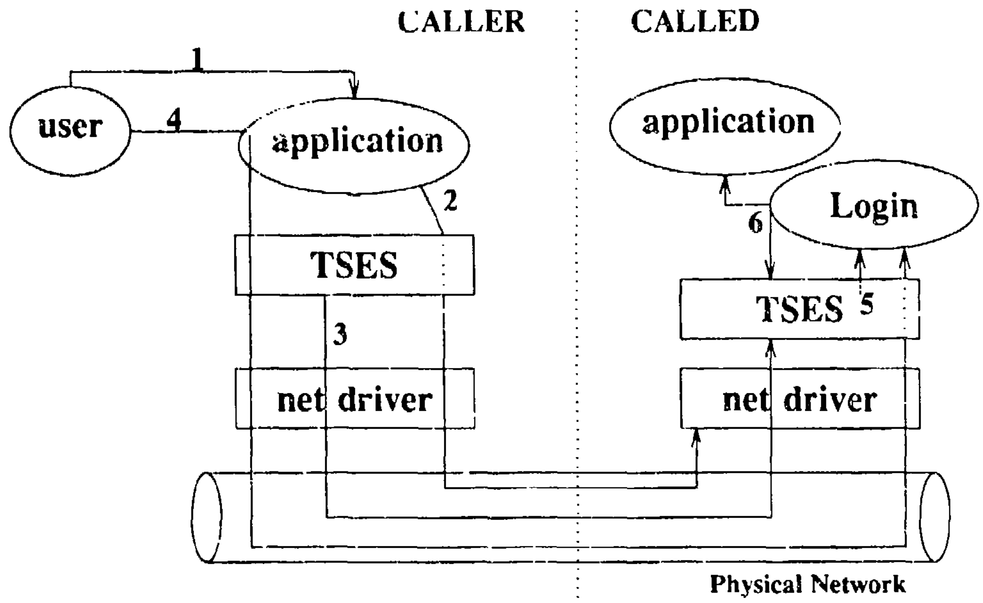
This section discusses the actual mechanisms used to implement the design. Figure 1 depicts the data flow between TSES modules during a remote login.

Chat - To be transparent, TSES must allow the caller uninhibited access to the network so that it can establish a connection to the called side. TSES assumes the network is secure; hence it does not interfere in communications between the caller and the network.

At some point, the caller TSES must transmit the caller's label. If it sends the label before the connection is made to the other TSES, it may interfere with the connection setup and/or the label may be lost. For this reason, the called TSES sends a unique string of characters back to the caller TSES to indicate that the connection has been made. The caller TSES looks for this string in all of the reads it performs on behalf of the application. When this string is encountered, the caller TSES sends the

[4] UNIX limitation

[5] STREAMS is a AT&T UNIX mechanism that allows stacking of protocol modules within the kernel



1. user invokes application
2. caller application establishes connection to other side
3. caller TSES passes label to called TSES
4. user logs into called host
5. login retrieves caller label from TSES
6. login labels TSES port and application

Figure 1: Data Flow During a Remote Log-in

caller's label to the called TSES. The exact string can be set by the system administrator. The string can be as long as desired. It should be a string that is guaranteed not to come from the network during connection setup. This restriction only applies during connection setup; there is no restriction placed upon data once the connection is made.

Spoof protection - As already mentioned, the caller TSES allows the caller unrestricted access to the network. We guarantee that the label read by the called TSES is not a phony label from the caller application by having the caller TSES tag the bona-fide label with a unique string that could not have come from the application. The called TSES is hence assured of its authenticity. This is very similar to the situation in which we had to let the caller TSES know that the called TSES had been reached by returning a unique string. However, here there is the possibility that the caller application will try to spoof this unique string. Therefore the caller TSES checks data written by the caller application to be sure it doesn't send out this string. As before, this restriction only applies during connection setup, and thus is quite acceptable.

Trusted listener to enforce label - TSES must prevent communications between the caller and the called application until the called's label is changed to match the caller's label. The chat ensures that immediately after the network connection is established, the called TSES has the caller's label. *Login* restricts the session to this label before turning the connection over to the application server. System V/MLS uses a sessions database file to control the range of labels between which a subject can switch.[6]

[6] System V/MLS supports dynamic relabeling of subjects. In other words, a user, can change his operating label from UNCLASSIFIED to SECRET without logging off the system (as long as the user has the appropriate clearance).

If *login* detects a TSES port beneath it, it sets the session minimum and maximum label to that of the caller. Also, the network port (i.e., the called TSES port), is labeled with the caller's label. This doubly ensures that under no circumstance, can the called application ever change its label and violate the security policy. It also protects against other processes sending and receiving data from this port if not properly labeled. System V/MLS provided easy-to-use mechanisms for setting session limits and for labeling devices.

Auditing - System V/MLS provides an extensive security audit trail (SAT). TSES has added its own probe point to the SAT driver. It is used to record network accesses on the called host. In particular, the caller's real user ID, process ID, and *hostname* are recorded. This allows remote activity to be traced back to the point of origination, when the audit trails of the caller and called hosts are analyzed.

Label mapping - The exact mechanism used to support network labels is an extension of the standard (non-networking), System V/MLS labels file. A new element, the network label, is added to the label structure. The labels file-searching library routines are enhanced to handle this new field.

TPATH (TRUSTED PATH)

One of the goals of our design was not to have to trust network applications. Unfortunately when a user logs into a remote host, the password must pass through the untrusted application (e.g., telnet, cu). It would be a simple matter for the application to steal the user's password and pass it on to another user. This other user could then log onto the remote host and obtain all information authorized for the original user. This would circumvent mandatory access controls and render useless all identification and authentication measures. An alternate data path is needed for this sensitive information. The result is a network trusted path mechanism for use during remote logins.[7] A detailed description of the process used to implement the trusted path is shown in Figure 2.

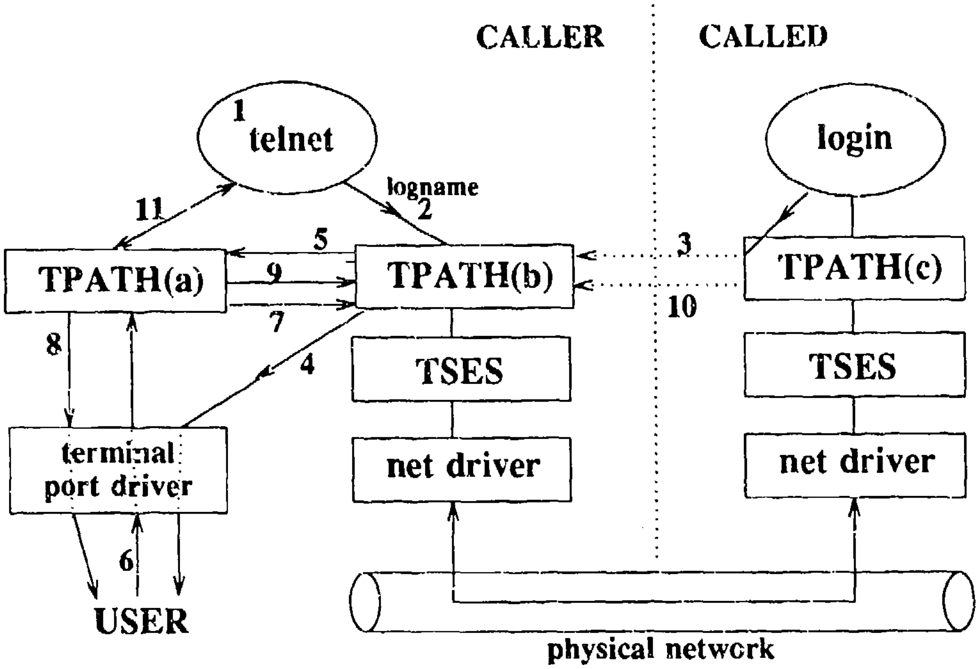
High Level Issues

A B2 trusted path, as defined by the Orange Book (i.e., for stand-alone systems), requires that only trusted software may exist between the user and the TCB for initial log-in and authentication. System V/MLS provides a B2 trusted path capability. However, in network configurations, a user may perform a remote login any time after initial log-in to the local host. If a trusted path is required between a user and a remote TCB, that implies a trusted path through the local TCB. The user must be unambiguously assured that this trusted path has been established. We exploit the fact that the user's terminal is directly connected to the kernel (hence the TCB) of the local host for this assurance.

Initiation of the trusted path - We had the choice of letting the user initiate the trusted path or making the *login* process initiate it. We chose to let *login* initiate the trusted path to make it easier for the user and to ensure that the user could not bypass the mechanism.

Data Transparency - A special string must be sent over the network from *login* to request set up of the trusted path. We call this string the Request Trusted Path string (RTP). The RTP string does not have to be a secret. However, it needs to be something that can only be sent by a trusted process such as *login*. If anyone could send this string, then a user could be spoofed into believing he is talking with a trusted process like *login*. To ensure a non-trusted process cannot send this string across the network, we use a process known as "data stuffing"[5]. Suppose the RTP string is "1234" and an untrusted

[7] As it turned out, the resulting capability can be used for a B3 trusted path within the local system or across a network



1. The user executes telnet which sets up a connection with the remote host
2. login prompts for logname and the user reponds
3. login signals TPATH(c) to set up a trusted path. TPATH(c) sends the RTP string to TPATH(b)
4. TPATH(b) receives the RTP string and prompts the user for the ATP string
5. TPATH(b) signals TPATH(a) that the trusted path is set up and waits for TPATH(a) to signal that it is ready
6. The user enters the ATP string and TPATH(a) catches it
7. TPATH(a) checks to see if the trusted path is set up and signals TPATH(b) that it is ready
8. TPATH(a) informs the user that the trusted path is set up
9. login requests the password from the user. The password is passed along the trusted path in the kernel
10. login sends the RTP string to cause takedown of the trusted path
11. All remaining communication goes through telnet as normal

Figure 2: Data Flow During Trusted Path Setup and Takedown

process needs to send the string across the network as part of the session. The string will pass through the kernel on the remote host. After seeing "123", the kernel inserts (stuffs) a "0". The string goes across the network as "12304" which will not be interpreted as the RTP string. When the kernel on the local host sees the "123" it checks the next character. If it is a "4" it initiates a trusted path. If, as in this example, it is a "0", it will strip the "0" from the string and send it on to the user. This technique effectively allows in-band signaling that is completely transparent to applications.

<u>Setup assurance</u> - The user must be assured that a trusted path has been set up. For example, a trojan horse program on the user's host could make the user believe a remote connection was made and request the user's password, masquerading as the *login* process. Since the real *login* never got executed, the trusted path was not set up. The method used to assure the user of a trusted path had to be non-spoofable. We explored two alternatives:

1. After the trusted path is set up the kernel sends a special string (password) to the user to prove to the user the trusted path has been set up. This password is known only by the user and the kernel. This is referred to as the "kernel password" method. The password would be randomly generated by the kernel and given to the user at initial log-in to the local host. This method required the user to remember this password throughout the session.
2. Login initiates the trusted path, however the user is requested to enter a special string to authenticate set up of the trusted path. We refer to this string as the Authenticate Trusted Path (ATP) string. The ATP string is caught by the kernel which checks to ensure a trusted path has been set up. If not, it then returns a warning message to the user. If a trusted path has been set up, it confirms this fact to the user and enables the trusted path.

The second method was chosen due to its cleaner user interface. Figure 2 presents a more detailed description of the process used to set up and takedown the trusted path.

Because the user's terminal is connected directly to the kernel of the local host, we can be assured that nothing can interfere with the transmission of characters from the terminal to TPATH. When TPATH sees the ATP, it informs the user as whether a trusted path has been established. The ATP string used to enable the trusted path is the same for all users and can be publicly known. The ATP string should be short and something the user does not frequently enter during a login session. This type of limitation is not unusual; most network connection processes have an escape sequence (e.g., "~." for *cu* and "~q" for telnet). Using this method, even if the user enters the ATP string during a session, the worst that can happen is that the user will get a warning message from the kernel that a trusted path has not been set up.

<u>Multi-Hop</u> - This design accommodates multi-hop network sessions. Multi-hop is the capability to log-in on a remote host and then from there log-in on yet another host. The only additional TPATH functionality needed is for any kernel that is acting as an intermediary to just set up the trusted path and pass on the RTP string to the next host. No special ATP string is needed on the intermediate systems.

<u>Trusted Path Takedown</u> - Takedown of the trusted path is also initiated by *login*. A second RTP string is used to signal the TPATH modules to take down the trusted path. No ATP string is needed from the user.

<u>Mechanism</u>

Figure 2 provides a description of the design implementation. TPATH is the name of the kernel module that provides the trusted path. A TPATH module is required over all user ports[8] and all network ports. We use *telnet* in our examples; however any network access process that connects with *login* at the remote host is valid. In the example, the lower case letters a, b, and c are used to differentiate between TPATH modules.

[8] A user port is the connection to the user's terminal. We are currently assuming that all users are connected to the network through a System V/MLS host and there are no terminals connected directly to the network.

TSTU (TRUSTED STU-III INTERFACE)

They are a number of methods for providing the underlying services that TSES requires. In this section, we describe one solution that provides connectivity between MLS hosts across the public-switched telephone network. This solution incorporates separate STU-III (Secure Telephone Unit) hardware in conjunction with host-resident TSTU software. We briefly describe the design and features of the STU-III product before discussing the overall solution in more detail.

STU-III (Secure Telephone Unit)

The AT&T STU-III is a communications terminal capable of transmitting voice or data, in either clear or encrypted form, across a phone line. It is a government-approved, unclassified (when not keyed) terminal that is designed to sit on a desk top. It is similar in appearance to a standard telephone. The unit contains an intelligent modem capable of transmitting information at 2400 or 4800 baud and a COMSEC module that performs the necessary encryption for secure voice and data transmission. Keying information for encryption is supplied by a "Crypto-Ignition Key" (CIK). This is a small, key-like device that must be inserted in the STU-III unit to enable operation in the secure mode. The key contains memory and supplies encryption information to the STU-III COMSEC module.

Two STU-III features essential to the TSTU development effort are Remote Operation mode and Remote Authentication.[9] The first enables the STU-III to be controlled remotely by a computer through an RS-232 connection. When the STU-III is configured in this mode, it can optionally be configured to provide the second essential feature, remote authentication. Each STU-III CIK contains authentication information which includes a key identification number. If remote authentication is enabled, this authentication information is exchanged between the STU-III's during secure call setup. The calling and called users are then optionally able to examine this information and determine whether to accept or deny the call.

TSTU Design Concepts

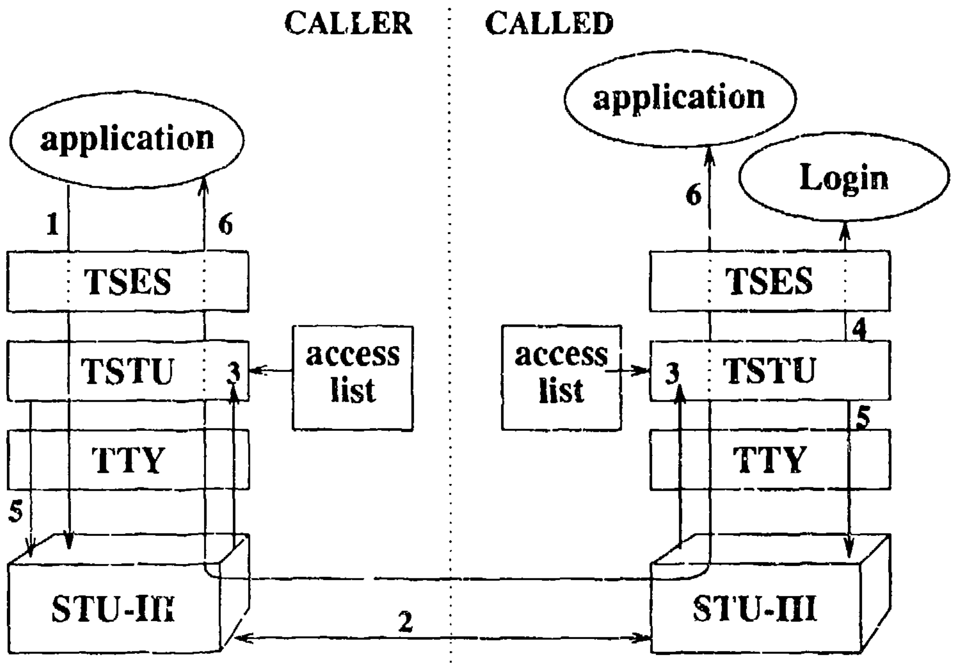
The TSTU module is designed to work in conjunction with the STU-III communications terminal to provide the set of network services required by TSES for secure operation. TSTU sits beneath TSES in the software stack and handles STU-III interface and control functions while assuring that the TSES network assumptions are satisfied at all times. This functionality enables MLS hosts to establish secure TSES networking sessions over the public-switched telephone network. The connection can be host-to-host or terminal-to-host. In this paper, we emphasize the host-to-host capability.

Utilizing the TSTU/STU-III solution, the services of connection-oriented sessions, information integrity and information confidentiality are all ensured by the STU-III. Network access control is provided by TSTU based on authentication information received from the STU-III during secure call setup.

TSTU is implemented as a character-based pseudo-device driver that sits directly on top of the standard character device driver associated with a STU-III serial port. It is designed to be transparent from above (to the user, and the TSES module), and it is identical on the caller and called side. The data flow occurring during the various stages of a secure STU-III network session is shown in figure 3. We refer to this figure throughout the TSTU design description.

Network access control - During secure call setup, TSTU (on both systems) provides network access control by examining the remote authentication information received from the STU-III. This information is compared to an access control list which is maintained by the system security administrator (step 4 in figure 2). After examining the remote authentication information, if TSTU determines that an illegal

[9] Only the AT&T STU-III provides this feature.



1. User process writes call request to the STU-III
2. STU-IIIs exchange crypto and auth information
3. TSTU get info from STU and compares to access list
4. Called TSTU passes label to login process
5. TSTU's write confirmation to the STU-IIIs
6. Call is established

Figure 3: Data Flow During a STU-III Network Session

party (as defined by the security administrator) is attempting to access the MLS host, it denies the call by sending the appropriate message to the STU-III.

The access control list is edited by the system security administrator and downloaded into the kernel. The list can be configured as a "good-guy" list, in which calls can be established only to those STU-III's using CIK's that are specifically included in the list. Alternatively, it can be used as a "bad-guy" list in which only invalid CIK identification numbers are listed. If TSTU detects that an attempt is being made to establish a call involving an invalid CIK, TSTU denies the connection. This flexibility enables each system security administrator to easily define a network access control policy that is consistent with TSES requirements.

Auditing TSTU may deny access based on the access control list. Denying access is a security relevant event which must be audited. TSTU records failed accesses in the standard System V/MLS audit trail. Successful accesses will be audited by TSES and by *login* and are not audited by TSTU.

Network security label - In addition to network access control, the authentication information embedded within the CIK's forms a basis for determining the network security label for the call. Each CIK is assigned a security label at its creation. During call setup, the security label of the CIK at each end of the call is received by the TSTU module within the caller and called MLS hosts. At each end of the call, the TSTU module calculates the lower of the two security labels and instructs the STU-III to establish the call at that level.

After the network security label for a call is determined, the TSTU module on the called side translates the network label into a host-specific security label and passes the label to the trusted System V/MLS *login* program, which ensures that the session label does not exceed the network label for the call (step 5 in figure 2).

Securing the STU-III command interface - The TSTU approach is to prevent the user from ever gaining free access to the STU-III while it is in command mode (i.e. accepting commands from the controlling computer, as opposed to transmitting the information over the phone line). When a user opens the port to the STU-III the TSTU software pumps setup information into the STU-III to ensure it is properly and securely configured. One of the pump commands turns off the capability that allows a user to escape from data mode to command mode by using a special escape sequence. The user only has access to command mode during call setup. Therefore, during call setup, TSTU monitors the information written by the user process to assure that it is consistent with the valid call request command format (step 2 in figure 3). This prevents the user from configuring the STU-III in an insecure mode.

CONCLUSIONS

We have found that it is possible to design a simple network interface that provides security at the B1 level as defined in the Red Book. Our solution enables System V/MLS to be used in a networked configuration. Existing applications and networks can be used without modification. Our strategy has been to build a security wall around existing networking applications so that they do not have to be trusted thus minimizing the addition of software to the TCB.

We have successfully prototyped this design over several networks including: STU-III, Ethernet*, direct-connect, and Datakit™. The prototype employed several network applications such as: telnet, UNIX mail, and UUCP. We are continuing work in this project particularly in the area of host authentication, so that we may remove the requirement for a closed network.

REFERENCES

1. *AT&T STU-III User's Manual*, AT&T, April 1, 1988
2. C.W. Flink and J.D. Weiss, "System V/MLS Labeling and Mandatory Policy Alternatives", Proceedings of the 1989 Winter USENIX Conference, February, 1989.
3. Douglas Comer, "Internetworking With TCP/IP", Prentice Hall, 1988.
4. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", EDS-TR-75-306, The MITRE Corp., March 1976.
5. A. S. Tanenbaum, "Computer Networks", Prentice Hall, 2nd edition, 1988

* Ethernet is a trademark of the Xerox Corporation

ON THE NEED FOR A THIRD FORM OF ACCESS CONTROL

Richard Graubart
The MITRE Corporation
Bedford, MA

ABSTRACT

The premise of this paper is that there are some access control policies employed in the DoD/Intelligence people-paper world which when mapped to the ADP environment cannot be adequately handled by the two traditional access control policies, mandatory access control (MAC) and discretionary access control (DAC). In this paper we will reexamin› the traditional access control policies. Then we will discuss one example of a policy that exists in the people paper world which is not adequately handled in ADP systems by MAC and DAC. Finally, we will propose one possible sol' tion to this problem in the form of a new type of access control policy.

MAC AND DAC REVISITED

MAC

MAC is defined in the TCSEC [DOD85] as "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (e.g., clearance) of subjects to access information of such sensitivity." The TCSEC further goes on to state the following conditions that must exist ın subject-object MAC relationship.

A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in ı e subject's security level include all the non-hierarchical categories in the object's security level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level.

In further examining MAC we can observe that MAC policies have three general attributes associated with them. First, MAC policies define a relationship between a subject and a object which is not changeable by the owner of the object. Second, when a subject reads an object and copies its content to a second object, the MAC restrictions imposed upon the first object propagate to the second object. Finally, MAC policies are uniform across all subjects and objects and are not tailorable on a subject/object basis. That is to say, if MAC prohibits a subject from accessing a object of specific sensitivity level, then that subject will be prevented from accessing all objects of that specific sensitivity level.

DAC

DAC is defined in the TCSEC as "a means of restricting access to objects based on the identity of subject and/or groups to which which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)". As with MAC, DAC also has three general attributes associated with it. First, DAC policies define a relationship between a subject and a object which is changeable by some authorized subject (i.e., the owner of the object). Second, when a subject reads an object and copies its content to a second object, the DAC restrictions imposed upon the first object do not propagate to the second object. Finally, DAC policies are NOT uniform across all subjects and objects. Rather they are tailorable on a subject/object basis. That is to say, although DAC may prohibit a subject from accessing an object of specific sensitivity level, DAC will not necessarily prevent that subject from accessing other objects of that specific sensitivity level.

THE TROUBLING CASE OF ORCON

We submit that there are applications in the people-paper world which use a form of access control which is not readily mappable in the ADP world to either MAC or DAC or a combination of the two. There are many examples of such applications, but for the purpose of this paper we will limit ourselves to the case of ORCON.

ORCON stands for ORiginator CONtrolled. In the people paper world, an individual receiving a document marked ORCON can only pass it on to another individual with the permission of the originator of the document. For the people paper world this policy is workable as one trusts people not to release documents which they are told are not releasable. The question for us is how we deal with this policy in the ADP world where we have subjects that act on the behalf of people, but unlike people are not trusted.

Let us take the following example. Subject x acting on the behalf of organization X marks object A ORCON, indicating that it can be released to subjects acting on the behalf of organization Y, but that the object is not releasable to subjects acting on the behalf of other organizations without the permission of the originator X. Moreover, any copies of A made by y (a subject acting on the behalf of Y) would be subject to the same policy restriction.

Traditional implementations of DAC would be inadequate for handling this situation. Under traditional DAC controls, subject y could read object A, and copy its content into a new object (C). The access control restrictions placed upon object C would be at the choice of subject y, the owner ot object C. The dissemination controls set by the subjects acting for X would be lost. [1]

MAC would appear to be more adequate for this task. A separate category could be associated with object A and with the subjects x and y. If y were to read object A and copy the data into some other object C, MAC policy would ensure that object C would also be labeled with the category. This would prevent subject y from arbitrarily giving subject some third subject z (acting on behalf of organization Z) access to object C. Thus, in this particular instance MAC is adequate for handling ORCON.

Now let us assume that the situation changes and that a new organization W (with subject w acting on its behalf) wishes to provide data in object B to organization Y but not to be shared with X or Z. For multiple reasons, the same category that was used to protect object A cannot be used to protect object B Using the same category would give subject x access to object B, and that is not acceptable. Also, the use of the same category would make it difficult to discern who was the originator of the data. It would appear that the solution would be to use another category and associate it with object B and subject w and y. However, while this solution works here, it is not a general purpose solution.

As the number of originators and recipients rises, so does the number of categories required to support the isolation of data. Each originator wishes to maintain exclusive control over their ORCON data. To accomplish this goal it would appear that for every object transmitted between an originator and a set of recipients, a separate category is required. Clearly it is possible (indeed likely) that the number of categories required could easily grow to a number that exceed the number supported by typical secure systems. Indeed the author knows of environments that process thousands of flavors of ORCON. Under the approach just described thousands of categories would be required to support such environments, and this would not be workable under most secure systems.

[1] Understanding the distinction between owner and originator is essential for understanding the ORCON problem. We use the term owner to refer to the individual or subject acting on behalf of an individual that is responsible for the creation of an object and is thus authorized to change DAC permissions on the object. In contrast to the owner, who is associated with an object, an originator is associated with the data contained in an object. The originator is responsible for the data, and for determining to whom the data can be released. This responsibility is true regardless of which object or objects contain the data.

The potential explosion in the number of categories is not the only reason that categories do not lend themselves to addressing ORCON. In the people-paper world, individuals are briefed into categories. Categories are used to represent a formal "need-to-know" whose characteristics are uniform in their meaning and restrictions based on policy set at the national level. For this reason categories associated with data (and the associated clearance on people) are accepted across DoD and intelligence organizations. The same is not true of ORCON. For ORCON there is no central 'clearing house' or guidebook that determines which category should be associated with a particular type of data, thus indicating which user should get access to the data. Categories corresponding to formal need-to know cannot be assigned to a subject if the user on whose behalf that subject is functioning has not been briefed into that category. In the case of ORCON, it is strictly up to the originator of the data to decide who has "need-to-know" for the data. Indeed it could be argued, that by using the MAC categories to address both formal need-to-know (for which the MAC categories were designed) and the more ad-hoc ORCON need-to-know, one is actually corrupting the use of the MAC categories. At the very least, for category based ORCON to be viable, it would be necessary for the system to note whether the category was based on formal need-to-know or ad-hoc need-to-know, and in the case of the latter it would also be necessary to note the originator.

SUMMARY

DAC is clearly inadequate for addressing ORCON as the access restrictions imposed by the originator would not propagate to new objects. MAC may be an acceptable solution to ORCON when a very limited number of ORCON 'flavors' are involved. However, the large number of ORCON flavors required by many applications would quickly exhaust the available number of categories on most secure systems. Even if the number of ORCON categories required was not an issue, MAC based categories do not support any mechanism that would allow for the association of an originator with the category. Therefore, MAC is not an acceptable general purpose solution to the problem of ORCON.

In short, neither MAC nor DAC adequately address ORCON in the general case.

POSSIBLE SOLUTION

From the previous discussion we can define the access control needs for ORCON is follows. First, ORCON requires that the access control relationship between a subject and a object is not changeable by the owner of the object (the same as in MAC). Second, when a subject reads an object and copies its content to a second object, the access control restrictions imposed upon the first object propagate to the second object (the same as in MAC). Finally, the access control restrictions are not uniform across all subjects and objects. Instead they are tailorable on a subject/object basis (the same as in DAC). Thus, what is needed for ORCON is a policy that has two of the characteristics of MAC, and one of those of DAC.

We submit as a solution a third form of access control which we call Propagated Access Control (PAC). PAC shares some of the characteristics of both MAC and DAC. As with DAC, PAC may be maintained in list form in a PAC list or PACL. PACLs (like ACLs) are associated with objects independent of the sensitivity label associated with the object. Thus, like DAC, PAC is tailorable on a subject/object basis. Unlike ACLs which indicate read and write access (among others), PACLs only are used to indicate read access. This is reasonable, as the ORCON problem is one of uncontrolled reading.

Another difference between DAC and PAC is that the only user authorized to change a PACL is the originator of the PACL, not the owner of the object with which the PACL is associated. In order to provide the originator the ability to change the PACL of an object, the identity of the originator must be associated with the object.

The most important trait of PA 'Ls (and their associated access control restrictions) is that PACLs propagat. to new objects. Whenever an authorized subject reads an object, the PACL of the object becomes associated with the subject. Any new object created by the subject acquires the PACL of the subject. In this regard PACLs resemble the floating information labels of the Compartmented Mode Workstations (CMWs) [WOOD87] which also tend to propagate. Note that this illustrates two additional differences between PACLs and traditional ACLs. First, ACLs are only associated with objects. PACLs are associated with subjects and objects. Second, PACLs propagate along to objects and subjects. Traditional ACLs do not propagate. PACLs are maintained with subjects and objects so long as the subject/object contains data.

ILLUSTRATIVE EXAMPLE

The use of PACLs may be better understood through example. Originator X (via its surrogate subject x) creates object A and associates a PACL with object A. The PACL indicates that X is the originator of the PACL and that only subject y (surrogate for recipient Y) can read the object. Subject y reads the object and in so doing the PACL becomes associated with subject y. If subject y creates a new object C, then the PACL becomes associated with object C. While subject y may be the owner of object C, it is not the originator of the PACL. As such, it cannot change the PACL, and thus cannot give any other subject read access to object C. However, because y is the owner of object C, y can still impose additional DAC-based read restrictions on object C as well as DAC-based write restrictions.

Combining PACLs

Note that PACLs may be combined. Let us continue with the current example, but now let us also assume that some subject w (working on behalf of Originator W) creates object B and associates a PACL with object B. This PACL indicates that only subjects y and z can read object B. To avoid confusion we will refer to this PACL as PACL_W (and the PACL associated with object A is PACL_X). Subject y, which has already read object A, now reads object B. Normally PACL_W would become associated with subject y. However, PACL_X is already associated with subject y. Therefore, the two PACLs become AND-ed together and the resultant PACL (call it PACL_XW) is associated with subject y. PACL_XW consists of those subjects which are common to both PACL_X and PACL_W (in this case subject y) and lists both originators (in this case W and X). Any object subsequently created by subject y will have PACL_XW associated with it. This is reasonable, as subject y contains ORCON data from both X and W, and therefore the permission of both subjects (originators) is needed to release the data to any new subject.

Resetting PACLs

One consequence of the PAC mechanism is that all subsequent objects created by the subject acquire the subject's PACL. While this approach is secure it may be operationally undesirable in cases where the subject is some general purpose process (e.g., editor). What we really want is for the PACLs to be propagated to new objects only so long as the subject contains the ORCON data that caused the PACL to be set. We believe that one possible way that this result can be achieved is if the PACLs are implemented on a UNIX-based CMW.

Under UNIX, subjects, which are represented by processes, are given life by the fork and exec commands. The fork command creates a new process from an originating (parent) process, and that new process is a duplicate of the original process. The exec command purges the new process's address space (eliminating all of the process's memory), and replaces it with that of a new code body specified by the exec. The PACL associated with the process represents the data currently in the process's address space. When the process space is purged (by the exec) the associated PACL is replaced with PACL associated with the code body specified in the exec. If the PACL of the exec code body is null, then the PACL of the new process is set to null. Note that the scenario just outlined is similar to how the CMW handles floating labels when processes are forked and exec-ed. The PACL of a process may also be reset by the intervention of a authorized user who is privileged to rest the process's PACL.

LIMITS OF PAC

We make no claim that PAC is as strong as MAC. Given the nature of the algorithms involved in calculating PAC, we readily accept the argument that it lacks the mathematical simplicity of MAC. However, the non-propagation characteristics of PAC would appear to make it far more secure than traditional DAC. Nor do we doubt that PAC may be subject to a variety of covert channel threats (though we have no idea as to how large the bandwidth of such covert channels might be).

In some ways PAC is more like MAC than DAC. In particular, PACLs should be associated with storage objects (as are MAC sensitivity labels) rather than named objects (as are DAC ACLs). The reason for this restriction is that named objects (unlike storage objects) are allowed to overlap (e.g., views in a DBMS). This is not acceptable for PAC, as such overlapping may result in unauthorized data flows.

We do not suggest that PAC should be a replacement for traditional DAC. We believe DAC is the appropriate access control system for addressing privacy needs that arise in the DoD and non-DoD world. For those systems that require the enforcement of ORCON (or ORCON-like functions), PAC is a useful and needed supplement to DAC and MAC. In such systems, access by a subject to an object would require that MAC, PAC, and DAC restrictions all be passed before access is granted.

OTHER APPLICATIONS OF PACLS

We have used ORCON as an example of where a concept such as PACLs would be useful. However, addressing the ORCON problem is not the only utility of PACLs. PACLs could also be employed for handling release markings. Release markings are markings that often appear on printed output in addition to sensitivity labels (categories and hierarchical levels). Release markings indicate to which countries (other than the US) documents can be released. Thus, a document marked TS REL UK can only be released to a TS cleared individual who is either a member of US or UK. Similarly a document marked TS REL ROK can only be released to TS clear person who is either a member of the US or a member of the Republic of Korea.

As with ORCON, only the originator of the data in the document can determine to which nation the document can be released (the originator can also authorize the release of the document to a foreign national not covered by the release marking). Release markings, like ORCON, do not lend themselves to being supported by traditional MAC categories. As with ORCON there is the concern about rapidly multiplying of needed categories to deal with the various flavors of release markings. Another problem with applying traditional MAC categories to release markings is that release markings do not combine in the

REFERENCES

DOD85 *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December, 1985.

WOOD87 Woodward, J.P.L., "Exploiting the Dual Nature of Sensitivity Labels," 1987 IEEE Symposium on Security and Privacy, Oakland, CA, 1987.

same way as categories. When categories combine the result is more restrictive set of categories (e.g., AB is more restrictive than A or B). However, as release markings combine the result is less restrictive (REL UK ROK is less restrictive than REL ROK or REL UK).

We also believe that PACLs may be a useful means for addressing the virus threat. Viruses tend to propogate whenever the code body containing the virus is read or executed. An object with a PACL associated with it cannot be passed on to another subject (user) without the permission of the originator. Thus, a virus infected object which has a PACL associated with it, can only infect those user's processes that were originally given permission to access the object. While this will not stop viruses, it will slow down their propogation.

SUMMARY

This paper has shown that there exists at least one application in the people-paper world (i.e., ORCON) which uses a form of access control that is not readily mappable in the ADP world to MAC, DAC, or a combination of the two. The paper has demonstrated the inadequacies of attempting to employ DAC or MAC to solve the problem posed by ORCON. We have also proposed in this paper a third form of access control, PAC, which has the subject-object specificity of DAC, but the tight hold, and access propagation restrictions of MAC. PAC appears to be an ideal method of addressing ORCON.

This paper was originally spurred on by the Intelligence community's need to deal with ORCON. However, as we have noted, there are other applications in other branches of the DoD for which PAC is appropriate. As an example, we believe that PAC is an excellent tool for automating controls on release markings.

We note in passing that PAC may also be an effective way of dealing with proprietary data, especially if there are multiple contractors using the same system. For some limited applications it may be an appropriate way of curtailing the threat posed by Trojan horses and viruses.

Finally, we have shown how PAC dovetails nicely with the floating label concepts of the CMW. We believe that the CMW would make an excellent base to prototype PAC and demonstrate its feasibility.

The Digital Distributed System Security Architecture

Morrie Gasser, Andy Goldstein, Charlie Kaufman, Butler Lampson
Digital Equipment Corp.
85 Swanson Rd., Boxborough, Mass. 01719

Abstract

The Digital Distributed System Security Architecture is a comprehensive specification for security in a distributed system that employs state-of-the-art concepts to address the needs of both commercial and government environments. The architecture covers user and system authentication, mandatory and discretionary security, secure initialization and loading, and delegation in a general-purpose computing environment of heterogeneous systems where there are no central authorities, no global trust, and no central controls. The architecture prescribes a framework for all applications and operating systems currently available or to be developed. Because the distributed system is an open OSI environment, where functional interoperability only requires compliance with selected protocols needed by a given application, the architecture must be designed to securely support systems that do not implement or use any of the security services, while providing extensive additional security capabilities for those systems that choose to implement the architecture.

1 Overview

The state of the art of computer security today is such that reasonably secure standalone operating systems can be built, and reasonably secure connections between the systems can be implemented. The purpose of the Digital Distributed System Security Architecture is to permit otherwise secure standalone systems to interoperate in a distributed environment without reducing the level of security and assurance of those systems. By "interoperate" we mean the ability to use, in a distributed fashion, all of the security capabilities inherent in standalone systems. Users "login" just once to the distributed system, users and objects have unique global names, and mandatory and discretionary access will be enforced regardless of the relative locations of the subjects and objects.

This architecture primarily addresses features for "commercial-grade" security and lower TCSEC [DOD85] classes up through B1. It addresses many security needs outside the scope of the TCSEC, and does not cover assurance requirements required by TCSEC classes B2 through A1. However, nothing precludes a system from implementing this architecture with a level of assurance beyond B1.

The architecture makes extensive use of encryption. Confidentiality and integrity for communication using symmetric (secret) key cryptography is presumed to be inexpensive and pervasive. Asymmetric (public) key cryptography is used for key distribution, authentication and certification. Users authenticate themselves with smart cards containing private keys and mechanisms to calculate cryptographic algorithms, and all systems possess their own private keys to authenticate themselves to other systems.

Authentication is assisted by the use of certificates, digitally signed by certifying authorities and stored in a distributed naming service that provides a hierarchical name space. A certification hierarchy tied to the naming hierarchy, along with the use of certain naming conventions, eliminates the need for global trust or trust of the naming service. Systems that need to act on behalf of other systems or users are explicitly the right to do so through certificates signed by the delegating parties.

In this paper key terms defined here are in *italics*. While most of these terms are well-known, the definitions here may be unconventional, different from past usage in similar contexts.

2 Security policy and reference monitors

The traditional concept of a single security policy and reference monitor [Ames83] for the entire computer system is not practical for a distributed system. While there are certain distributed environments where security management responsibility is centralized, in most cases the individual systems comprising the distributed system must be considered to be independently managed and potentially hostile toward each other. Mutually suspicious systems should be required to cooperate only to the extent that they need each other's services, and no more. Moreover, even if we could assume that a large distributed system were centrally managed under a single security administrator, building a distributed reference monitor to provide all the security capabilities of a single system presents unsolved research challenges.

Rather than a common security policy and reference monitor, each system implements its own reference monitor enforcing its own policy. Each reference monitor is responsible for controlling access to the objects it maintains. In the most general case the reference monitor on one system receives a request to access one of its objects from a subject controlled by a reference monitor on another system. Access is permitted only if the reference monitor for the object can verify that proper subject authentication has taken place, that the system from which the request is received has been duly authorized by the subject to make that request, and that there is compatibility between the security policy of the requester's reference monitor and that of the object's so that access rights can be evaluated. Implicit in this compatibility is some level of mutual trust of the reference monitors.

In today's systems, reference monitors are usually operating systems and large subsystems or servers that manage their own objects directly. In the future distributed system any application may become the reference monitor for its own set of subject and objects. The subjects and objects controlled by such a reference monitor may be implemented out of other subjects and objects controlled by another underlying reference monitor. Also, in certain limited cases, several systems may "team up" to comprise a larger system implementing a single distributed reference monitor, all implementing exactly the same policy and fully trusting each other. At this time the security architecture does not explicitly address the mechanisms needed to construct composite objects or multiple reference monitors in a computer system, and does not impose any structure on the relationship between reference monitors. The architecture simply allows all reference monitors who are able to identify their own components to securely manage their globally accessible resources in a uniform manner.

For the most part, the architecture defines interoperable security mechanisms and does not address degrees of assurance of reference monitors as addressed in the TCSEC. Regardless of their assurance, it is expected that all systems conforming to the architecture will implement interoperable mechanisms. Assurance, where important, will impose design constraints and methodologies on individual systems but should not influence the security-related external behavior of those systems. For example, a security kernel architecture might permit a reference monitor to be contained within a subset of a whole operating system, allowing that system as a component of the distributed system to be granted an A1 rating. Such a subset must implement all of the relevant security mechanisms that might be implemented by other (e.g., C2) systems on the distributed system where the entire operating system acts as a reference monitor. The architecture also permits different reference monitors to have a mutual understanding of their respective degrees of assurance and accreditation ranges so that they can determine whether their security policies are compatible.

3 Computing model

The world is made up of interconnected systems and users. A system is comprised of hardware (e.g., a computer) and software (e.g., an operating system), and a system can support one or more software systems running on it. Systems implement other systems, so, for example, a computer implements an operating system which implements a database management system which implements a user query process. In this manner, a system whose reference monitor controls one set of objects might implement another system with a reference monitor for another set of objects. For purposes of the security architecture, we rarely distinguish between the different types of software systems such as hosts, operating systems, database management systems, nservers, and applications, and we rarely need to get involved in the possible hierarchical relationship between systems built out of underlying systems.

A user interacts physically through a keyboard and screen that are electrically (or securely) connected to a system: usually a workstation, timesharing system, or terminal server. The user invokes an operating system and applications processes on that system which he trusts to perform work on his behalf. The work may involve only data local to the workstation, or may involve data on and interaction with remote services on other systems.

All interactions, direct or indirect, between a user and a remote system pass through the user's local system. Therefore the local system must be trusted to accurately convey the user's commands to the remote system, and the remote system must be trusted to implement the commands. Because the local system has access to any remote information that the user can access on that remote system, the user has no choice but to trust his local system to be faithful to his wishes.

The remote system, in order to satisfy a user's command, may need to forward the command, or make an additional request, to a second remote system. In such a case the first remote system must also be trusted to accurately reflect the user's wishes. In general, the user may interact through a chain of systems, where the user must trust each system in the chain, and where communications between the systems in the chain is assumed to be secure so that the commands and responses are safe from alteration, forgery or disclosure.

4 Message authentication and secure channels

The architecture depends extensively on the use of a *message hash* function that yields a message authentication code (MAC), a short "digest" of a message that is much more efficient to communicate and store than the original message. A good hash function has the property that, given the MAC of one message, it is computationally infeasible to create another message having the same MAC. While cryptographic MACs are frequently used where two parties already have established a cryptographic association, message hashes of greatest interest to the architecture are those whose security does not depend on knowledge of shared keys, so that anyone can check the MAC of a message but nobody can forge another message with the same MAC. This permits MACs of widely used messages to be freely distributed without prior negotiation of keys. An example of such a hash function is provided in Annex D of the CCITT recommendation X.509 [CCITT88b].

In this architecture, *communicating securely* means satisfying one or both of the properties: (1) knowing who originally created a message you are reading, which we call *authentication*, and (2) knowing who can read a message you create, which we call *confidentiality*. The ISO (International Standards Organization) term "data origin authentication" [ISO88b] is equivalent to property (1). Our concept of authentication also implies "data integrity": assurance that the message you are reading is exactly the same as the one that was created (if the message is altered then it's not a message from the originator).

The term "peer entity authentication", used by ISO to describe the property that you know with whom you are communicating, is subsumed in our architecture by both properties (1) and (2). In the security architecture it is meaningless to have "peer entity authentication" by itself: without either confidentiality or data origin authentication (with integrity) you cannot tell whether your message is protected or whether you are actually receiving what was sent and so communication is not secure in any practical sense.

ISO's definition of "confidentiality" is also not strictly the same as ours, as we assume that the recipient is known and must therefore have been authenticated at some time in the past.

The concept of a *secure channel*, introduced by Birrell, et al. [Birrell86], is an abstract way of viewing how we accomplish properties (1) and (2). A channel is a path by which two or more entities communicate. A secure channel may be a protected physical path (e.g., a physical wire, a disk drive and medium) or an encrypted logical path. A channel need not be real time: a message written on a channel may not be read until sometime later. A secure channel provides either authentication or confidentiality, or both, while an insecure channel provides neither. Communication via insecure channels is permitted but is not addressed by the architecture.

Secure channels have identifiers known to the senders and the receivers. A secure physical channel is identified by a hardware address such as an I/O port number on a computer or a disk drive and block number. An encryption channel is identified by an encryption key. Any message encrypted under a given key is said to be written on the channel identified by that key, regardless of whether that message is "sent" anywhere. When the message is decrypted it is said to be read from the channel. The ciphertext of an encrypted message may be written on another channel before being decrypted: typically the ciphertext is written on an insecure channel for transmission, read from the insecure channel, and finally read from the secure channel by decryption.

For a secure channel that provides authentication, the senders are known to the receivers and are thus authenticated. Specifically, a receiver of a message on a secure channel can determine that the message was written by someone in a known set of senders. If there is more than one possible sender then, in order to determine the actual sender, the receiver must trust the senders to cooperate by properly identifying themselves within the text of the message or by not sending unless requested.

For a secure channel that provides confidentiality, the receivers are known to the senders and are authorized by the senders to receive the information. In most cases there is usually only one possible receiver. If there is more than one, and the sender wants to limit the message to a specific receiver, then the sender must trust the other receivers not to read messages unintended for them.

A symmetric key channel (identified by a secret encryption key) provides confidentiality and, can provide authentication with the use of a MAC for integrity. For a symmetric key channel all authorized senders and receivers must share the same key, and therefore all senders and receivers are in the set authorized to read or write information on the channel.

An asymmetric key channel (iden'ified by either its private or public key) provides authentication if a message is encrypted with the private key, or confidentiality if a message is encrypted with the public key. A single encryption operation cannot provide both properties (even though a single public/private key pair can provide both). Typically there is a unique pair of keys for each principal. The principal keeps its private key confidential and the public key is made generally available (online or through some directory service). This and the following description of asymmetric key channels primarily applies to the RSA public key algorithm [Rivest78].

In an asymmetric key channel used for authentication, the sender creates a "digital signature" of the message by encrypting the MAC of the message using the sender's private key, and sends the signature along with the original (plaintext) message. Any recipient who knows the sender's public key can verify the signature by recalculating the MAC and comparing it to the decrypted signature, to determine whether the original message was signed by the sender. The sender is authenticated to the receiver because only the sender knows the private key used to sign the MAC.

It is impractical for all entities in the distributed system to know the correct public keys of all other entities with which they want to communicate. Entities are typically identified using network addresses or names expressed as character strings. A special kind of signed message, termed a *certificate*, is used to unforgeably associate the name of an entity with its public key. Certificates have a number of related functions as well described below.

In an asymmetric key channel used for confidentiality, a sender encrypts a message with a receiver's public key which only the single receiver can decrypt with the private key. The sender's message is thus confidential. Since anyone can encrypt a message with someone's public key this channel does not provide authentication of the sender. To provide both authentication and confidentiality, a message must be first signed with the sender's private key and the result encrypted with the receiver's public key. In practice, both steps are rarely applied to the same message, and in fact the architecture rarely needs to make use of asymmetric key cryptography for confidentiality.

The most popular algorithm for symmetric key encryption is the Data Encryption Standard (DES). However, the DES algorithm is not specified by the architecture and, for export reasons, ability to use other algorithms is a requirement. The preferred algorithm for asymmetric key cryptography, and the only known algorithm with the properties required by the architecture, is RSA. As with DES, the architecture does not specify and will not depend on the details of the RSA algorithm; another algorithm with similar properties, if invented in the future, is permitted.

Access control does not apply to secure encryption channels: a secure encryption channel as defined in the architecture is created when needed and is not a limited resource or object to be protected. Access to the channel is determined by those who possess the encryption keys. A physical channel (whether or not it is used for security) is a limited resource to which access may need to be controlled. In such a case the channel would be treated as an object, with an ACL (see section 7) and perhaps mandatory access controls.

When two systems interact through a secure encryption channel (e.g., two nodes on different LANs using end-to-end encryption across a wide area network), there may be many intermediate systems (gateways, bridges or routers, etc.) in the path between the end systems. These intermediate systems are needed to

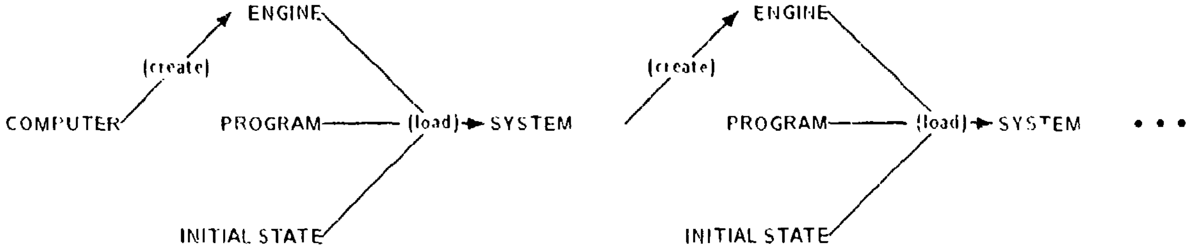


Figure 1: Computers, systems, programs and engines.

support communications for the applications in the end systems but need not be trusted to keep the channel secure. Intermediaries in a secure physical channel, on the other hand) must be trusted.

For some applications involving several systems there are a number of secure channels between pairs of systems participating in the application. For example, consider a user on a workstation who submits a query that gets forwarded to a remote DBMS which accesses a record in a file on a file server. In this example the DBMS system is an endpoint of one secure channel (from the workstation) and an originating point for a second secure channel (to the file server). Normally all three systems must be trusted by the user because the DBMS processes both the query and the results being returned and there is no secure channel directly from the user's workstation to the file server. On the other hand, if the file server encrypts a record that it hands to the DBMS, and the DBMS simply forwards the record to the user's workstation for decryption, then there is a secure channel between the file server and workstation and the user does not need to trust the DBMS to protect that record from disclosure.

In the context of communications it is simplest to think of secure channels as secure transport layer connections providing confidentiality and integrity of the data, even though transport is not the only place where there may be secure communications. In the context of authentication a secure channel is usually something defined by a given encryption key that is used to pass signed messages.

At this time, the architecture is not tied to any specific protocol suite. The detailed specifications of protocols, to be prepared eventually, will describe how to set up secure channels using specific network protocols.

5 Computers and loading

A *computer* is a system made up of a particular physical set of hardware components running some boot code. All connections between the computer and the rest of the world must be through secure channels.

An *engine* is a hardware or software device created by a *system* that can be *loaded* with a program to produce another system. The computer running its boot code provides an engine into which an operating system can be loaded, thereby creating what we commonly refer to as a host or *node*. Another example of an engine is a process provided by an operating system. When loaded with an application program, the running process becomes a system. These relationships are illustrated in figure 1.

A *specification* is a description of a system's behavior (e.g., the specific behavior of a VAX 6250 computer or that of VMS 5.0, documented in some manual). While a specification is rarely written down precisely, users of (or systems interacting with) a system that is "certified" to meet a given specification can be assured that the system will behave as they expect. The architecture deals with the problems of certifying a system and determining whether that certification was done by someone you trust. Certifying a system does not have anything to do with software correctness—certifying that a system meets the "VMS 5.0 specification" simply means knowing that a specific program (the "VMS 5.0 boot image") was loaded into a specific type of system (a "VAX computer") using specific sysgen parameters. It is assumed that the particular boot image does what is intended—proving that the program in fact meets some written specification is outside the scope of the architecture.

In general, software is certified by the system loading the engine it has created, by verifying that the

MAC of the software image is equal to the expected value for that software's specification. For example, if the MAC of an image you have just loaded is equal to the MAC you expect for "VMS 5.0 boot image" then you can be confident that you have just loaded a program that will behave according to the "VMS 5.0 specification." The MACs of various images that may be loaded are contained in certificates.

Each system, including the computer hardware itself, has a secret (the private portion of a private/public cryptographic key pair), generated randomly when the system is installed or created, which it uses to authenticate itself and to certify systems it creates. A system is responsible for protecting its secret from disclosure to the created systems. Through chains of reasoning beginning with the computer and ending with an application system (for example) it is possible to certify any desired aspect of a system or its behavior. In contrast to software systems' secrets which are created each time the system is rebooted, computer secrets are semi-permanent, stored in programmable read-only memory.

When a computer is asked to boot some software, the boot hardware in the computer (usually implemented as software in read-only memory) calculates a MAC of the operating system that it has loaded, and, before permitting execution, verifies (by checking certificates provided to it by system management) that an operating system with the designated MAC is permitted to run on that computer. If verified, the boot hardware generates a private/public key for use by the loaded operating system, signs, using its boot secret, a certificate associating the MAC with the public key, deletes the boot secret from any place that operating system can get to, and then begins execution of the loaded operating system. The operating system, in turn, uses its new private key as a secret to sign for other systems (applications) that it loads, and so on. When asked to authenticate itself to a remote system, the operating system presents as credentials its certificate signed by the computer. In this manner, with minimal new mechanisms in the hardware, the computer has protected itself from being loaded with malicious software, and other systems who trust the computer's boot hardware can verify the identity of the loaded operating system. Of course, if the operating system is compromised after it starts running nobody may find out.

6 Naming

A *principal* is an entity that can be granted access to objects or can make statements affecting access control decisions. Principals are subjects in the TCSEC sense, but not all subjects are principals. For example, a principal may spawn multiple process within a system, each one identified as its own subject to the operating system, but the architecture treats each of these subjects as if they were the original principal and makes no attempt to isolate them from each other. When a principal accesses an object the reference monitor for the principal in control of the object must have some way of identifying the requesting principal, and this identification is in the form of a unique global identifier. These global identifiers are Digital Naming Service (DNS) names.

Users and systems (nodes, servers, etc.) are named principals who have DNS names. There are also principals such as smart cards, processes, and sessions that do not have DNS names and which always act on behalf of other (named) principals. The use of DNS is pervasive in the architecture, but the primary reason for DNS names is so that users can identify principals and can enter their names on access control lists (see section 7). Without DNS names, users would have to identify principals with unwieldy cryptographic keys.

DNS has a hierarchical tree structure, with a single root at the top and directories at the branches. A principal's name lies within some directory and the principal always knows (or can determine) its place in the hierarchy from the root; the series of directory names from the root down to the principal is the principal's DNS name. In figure 2, for example, the full DNS name of principal P8 is TOP.MID-1.LOW.BOT.P8. While DNS names are human-readable, it is not expected that people will have to type a full DNS name very often. The DNS structure and the services provided by DNS are very similar to the directory proposed by CCITT and ISO [CCITT88a].

Principals, and even large sections of the hierarchy (subtrees), may be moved from one place in the tree to another as organizational and other associations change. This means that a principal's name (usually, just the directories in a principal's name) can change, perhaps without the principal's awareness. When a subtree is moved a *symbolic link* may be placed at the old location's parent directory that points to the new location of the subtree, thereby permitting principals to be found using their old names (see figure 3). Symbolic links serve a number of other purposes not related to security.

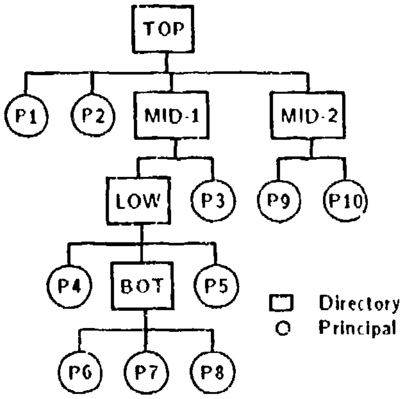


Figure 2: Example of DNS hierarchy.

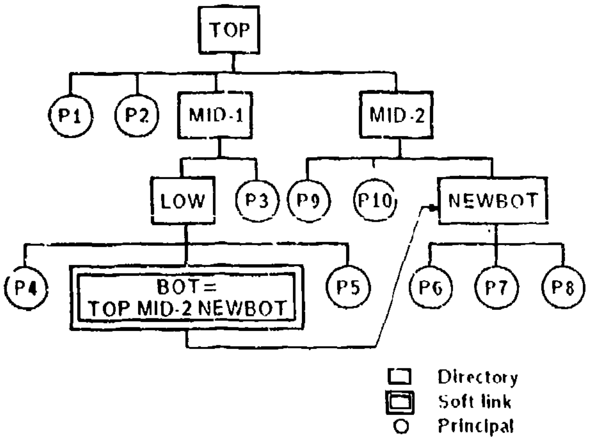


Figure 3: Symbolic link in DNS.

Because of symbolic links, a principal may be identified by several DNS names, only one of which is the true name. In figure 3, the principal originally known by the name TOP.MID-1.LOW.BOT.P8 in figure 2 is now located at TOP.MID-2.NEWBOT.P8, and may be referenced by either name due to the presence of the symbolic link at the old location of the BOT directory. To provide a fast way to determine whether two names refer to the same principal (something that the access control mechanism must be able to do) a principal also has a unique-identifier (UID) which doesn't change even if the DNS name of a principal changes. The UID is stored in DNS in the directory entry for the principal, and plays an additional role in the reassignment of names and definition of the directory hierarchy. With minor exceptions, the UID is used by the security architecture for performance rather than for security. Thus, the algorithm for enforcing uniqueness of UIDs is outside the architecture. In a few cases where security depends on uniqueness of UIDs, there are simple ways to enforce it.

Except for the names, UIDs and symbolic links, other aspects of the DNS architecture are not relevant to the security architecture and security (except certain types of revocation described in section 11) does not depend on correct functioning of the DNS servers. Of course, if DNS does not function correctly availability might suffer.

7 Access control

All information to which access is controlled is contained in objects. All objects have *access control lists* (ACLs): lists of principals (identified by DNS name) who may have access to the object, along with their access rights. There are a small number of architecturally defined access rights, such as "read," "write," etc., and some number of system-defined rights. It is the responsibility of the system (the reference monitor) controlling an object to enforce the ACL. An operating system, for example, enforces the ACLs for the files in its file system. The principal that controls an object is not listed on the ACL.

ACLs may contain names of *groups* of principals. Groups are objects with DNS names and may be created and modified by ordinary users, not just by system managers. All groups must exist as an explicit list of principals—there is no architectural support for "implicit" groups identified through some kind of naming convention (for example, "all principals contained in a given directory") but implementing such a capability is not precluded. However, large groups may be constructed out of smaller groups: groups may be nested (may name other groups) to an arbitrary depth. The ability to efficiently support both very small and very large groups, with tens of thousands of members, is essential for practical use of some of the security mechanisms specified by the architecture, and schemes have been developed that permit DNS to support them.

ACLs may list specific principals that are denied access, even if those principals are contained in groups that are permitted access. It is also possible to deny access to groups that are subgroups of other groups on the ACL. Certain other restricted forms of group denial are possible, but it is impractical, in a distributed environment where group nonmembership cannot be certified, to implement denial to arbitrary groups.

In addition to listing the principals that may access an object, the ACL may list the systems to which access may be delegated (see the discussion of delegation in section 10). This capability means that an object might not be accessible from "untrusted" workstations even if the user has delegated to that workstation.

ACLs may be implemented in a number of ways on different systems, but, because of their user visibility, it is important that ACLs have similar semantics on all systems. The VMS system-owner-group-world mask, or Unix owner/group/other bits, are primitive forms of ACLs, but such forms must be augmented (not necessarily replaced with something else) to provide the necessary semantics outlined above.

ACLs are objects themselves and have ACLs that specify who can read or modify them. An ACL may be its own ACL, or there may be other ACLs dedicated to ACL access. Figure 4 illustrates one way a file's ACL and an ACL's ACL may be related. In this figure the ACL for the ACL's ACL is itself.

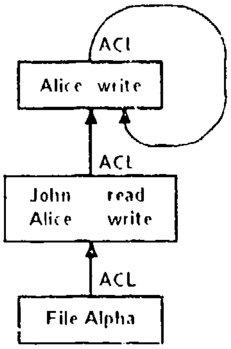


Figure 4: A file's ACL and an ACL's ACL.

8 Authentication

(In the following discussions we use as an example a principal sending a request to a system or service. In fact, the terms "system", "server" or "service" are just different names for principals —the model does not distinguish between a server and any other type of principal.)

In order to mediate access to an object that it controls, a server must *authenticate* that the identity of the requester is as claimed. Secure channels provide this "strong authentication." The password is the most common type of authentication mechanism used in systems today but the password does not provide a secure channel. At the beginning of a conversation, a set of messages are exchanged between a principal and a server, where the server establishes that it is in fact receiving messages from a secure asymmetric key encryption channel whose only possible sender is a given principal. Similarly, the principal may wish to mutually authenticate the server, and this is possible because the server is also a principal.

In order for a server to know that it is currently communicating with a given principal, a server must be sure that the signed messages it is receiving are not replays of old messages from a previous conversation (possibly sent by a third party). To deal with timeliness, a challenge/response scheme is used at the beginning of each conversation, where the server sends a random number to the principal and the principal returns the number in a signed message. Replay of a response to an old (different) challenge is not accepted. Within this signed message is other information that permits the two parties to continue to communicate in a manner that is safe from replays of past conversations.

Once two principals have authenticated each other using asymmetric key cryptography, one of them typically will generate a random secret key and send it to the other. This secret key will be used to communicate (using symmetric key cryptography) in a manner that provides continued authentication and confidentiality for future messages during the conversation. Symmetric key cryptography is usually used for data exchange because asymmetric key cryptography is too slow.

Authentication can also be initiated with symmetric key cryptography where a principal authenticates itself to a trusted online "key distribution center" and the key distribution center provides the information necessary for that principal to then authenticate itself to a server. The indirect authentication through a trusted third party is required because otherwise the server would have to be told the secret key of the principal, leaving the principal exposed to masquerading by the server.

Nodes and other systems that need to authenticate themselves have secret or private keys stored in nonvolatile memory within them, and they implement the RSA and DES algorithms using hardware or software. It is expected that software implementations of RSA or DES (without specialized hardware) will perform adequately for authentication at the beginnings of conversations, but specialized hardware will be needed to calculate DES at a speed adequate for data exchange. Before such specialized hardware becomes widely available, the authentication functions can be implemented in software without protecting the data exchange. This "authenticate at session initiation only" function provides some measure of security in certain applications even though the architecture does not recognize the subsequent unprotected data exchange as a security capability.[1]

Since users cannot remember RSA keys hundreds of bits long, and cannot calculate algorithms in their heads, user authentication requires a computer for the calculations and a portable means of storing the user's private key. Technology is just emerging that will provide both in the form of a "smart card". Each user possesses a smart card containing that user's private key, the user's secret personal identification number (PIN), and a microprocessor that can compute the RSA algorithm.[2] The user authenticates himself to the workstation by inserting the smart card into a reader, and entering the PIN into the reader (if the reader is trusted) or into the card (if the card has a keypad). The smart card refuses to operate if the correct PIN is not entered. The smart card then responds to a challenge from the workstation so that the workstation can authenticate the identity of the smart card. The workstation assumes that the user is in control of the smart card and thereby assumes it is communicating with the user through the keyboard and screen.

9 Certification

When an access request arrives at a server on a secure channel, that channel is usually unambiguously associated with the public key of the principal making the request.[3] However, access to objects is specified

[1] In some international applications data exchange can be authenticated but by law must not be encrypted. Authentication of data exchange requires the same high performance cryptographic hardware as does encrypted data exchange.

[2] There are smart cards that can do simple calculations and can store RSA private keys, but if the card cannot do the complete RSA calculation then the private key must be disclosed to some external device for the calculation. A smart card is much more secure if there is no function enabling the key to be read out.

[3] This explanation is greatly simplified; the association between a principal's public key and a given channel may be very indirect, involving many other secure channels and delegations.

in terms of DNS names on access control lists, not in terms of public keys, so just verifying the public key of the sender on a secure channel is insufficient for access control. To enforce the access control list the server must have some way to determine the DNS name that corresponds to that public key. To assist in this determination, the requesting principal provides its DNS name prior to the request, so the server's problem is to verify that the DNS name in fact belongs to that principal with the verified public key.

It is possible, but not practical, for each server to keep a table of DNS name-to-public key correspondence for all principals listed on its ACLs. A more general solution involves the use of certifying authorities (CAs) that are trusted by systems to provide this verification. A certifying authority is a principal that possesses its own private key, and its corresponding public key is made well known to the principals who choose to trust that CA. A CA willing to certify that a given public key belongs to a given DNS name signs a certificate stating that association. CAs perform other certifications as well (e.g., certifying that a given smart card's public key belongs to a user with a given DNS name, certifying that a given MAC identifies a given software image, and certifying that a given image may be loaded on a given computer), and CAs or other principals may also certify other things (such as group membership lists). In this section we are concerned only with the certification of a public key by a CA for use in authentication.

CAs do their certification as an offline process well in advance of the use of the certificates, usually when a principal's private and public key are first created. The mechanics of generating keys and becoming certified are details outside the scope of the architecture, but the process amounts to convincing a CA that the identity of a principal (e.g., its DNS name) corresponds to a given public key, in a manner similar to convincing a notary public of the correspondence between your identity and your signature. It is easy for a principal to prove, through a response to a challenge from a CA, that it possesses the private counterpart to an alleged public key, so the act of certification is one of verifying that the principal is in fact the one named.

Certification does not require that the CA either generate or know the private key of the principal being certified, so a principal does not expose itself to any threats if certified by an untrustworthy CA. A compromised CA only compromises those who trust its certificates.[4]

Any system that knows a CA's public key, and trusts the CA to vouch for the public key of the identified principal, can verify the signature on a certificate and can determine that the public key corresponds to the given DNS name. Certificates for authentication are usually stored in a DNS server, but a copy of the information (the name and public key, or perhaps the whole certificate), may be locally cached. While CAs may be online for convenience (e.g., to distribute newly signed certificates), CAs need not and in fact cannot work like online servers. Certification must involve an offline path to corroborate the identity of the principal.

By using signed certificates to determine public keys there need be no online "authentication server," and no centralized or replicated database of public keys is required (except to support revocation -see section 11). The certificates are distributed to the places where they are needed, and DNS provides a convenient mechanism for storing certificates locally.

There is no one CA that all principals are willing to trust for all authentications. Each directory in DNS has an associated CA (see figure 5), and several directories may share the same CA. Principals in a directory usually trust the directory's CA to certify other principals in that directory. The following lists the principals that the CAs in figure 5 are trusted to certify:

CA-TOP	certifies	P1, P2, P3, CA-BOT, CA-MID-2
CA-BOT	certifies	P4, P5, P6, P7, P8, CA-TOP
CA-MID-2	certifies	P9, P10, CA-TOP

CAs are also trusted by those principals to certify the CAs of directories immediately above and below them (but of course it is unnecessary for a CA to certify itself if that CA is also associated with an adjacent directory.)

Typically, principals trust CAs close to them in the hierarchy. A principal is less likely to trust CAs farther from it in the hierarchy, whether those CAs are above, below, or in entirely different branches of the tree. If a server at one point in the hierarchy wants to authenticate a principal elsewhere, and there is no

[4] When a server depending on a compromised CA manages the principal's resources or has been given the right to act on behalf of the certified principal (as when a file server manages a user's files or acts on behalf of a user) then the certified principal may be indirectly compromised

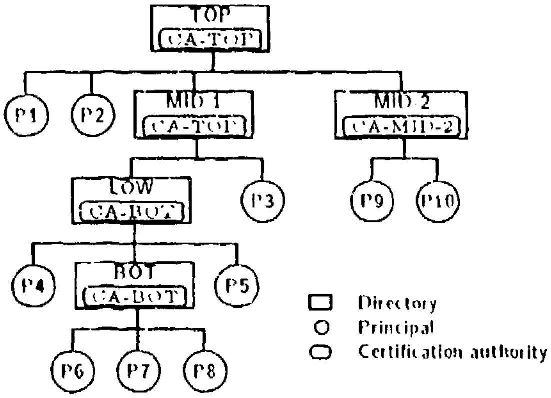


Figure 5: Certification authorities in directories of a DNS hierarchy.

one CA that can certify both, then the server must establish a chain of trust through multiple CAs. This chain involves all the CAs in the path from the server, up through the hierarchy to the first directory that is common to both the server and the principal ("leastt common ancestor"), and then down to the principal. For example, in figure 5, P7 can authenticate P5 by trusting only CA-BOT. If P7 wants to authenticate P10, then all three CAs in the figure must be trusted because the least common ancestor is CA-TOP.

The authentication process assumes that the principal is identified to the server by a full DNS name, and that the server can determine the "least common ancestor" and correct CA path by a simple comparison of its own name with that of the principal. (For example, the least common ancestor CA common to TOP.MID-1.LOW.BOT.P7 and TOP.MID-1.LOW.P5 is CA-BOT in TOP.MID-1.LOW.) By use of a symbolic link on one of the intermediate directories it is possible to establish a shorter path by making it appear that the server and principal lie in a common subtree below their least common ancestor. A symbolic link alone is just a pointer for convenience of lookup, but when augmented with a "certification cross link", the certification path reflects the symbolic link path. A certification cross link permits a CA at one point in the hierarchy to directly certify any other CA or principal, thereby eliminating one or more higher level CAs from the default chain of trust. A cross link is a certificate signed by a CA that provides the public key of the CA for the target directory (or principal), and states that the name translation specified in the corresponding symbolic link is correct.

In figure 6, the cross link at the symbolic link MID in directory LOW permits P7 to avoid having to trust CA-TOP to certify P10. Instead, P7 authenticates P10 by trusting CA BOT (to certify CA-MID-2), and CA-MID-2 (to certify P10). The least ancestor CA common to TOP.MID-1.LOW.BOT.P7 and TOP.MID-1.LOW.MID.P10 is CA-BOT in TOP.MID-1.LOW.

The hierarchical nature of the certification architecture described here is similar to that used in ISO's directory authentication framework [CCITT88b]. In ISO's architecture, however, users who have no a priori knowledge of the certification hierarchy must potentially trust all CAs because there is no explicit way to indicate the "least common ancestor" or other limitations to the chain of trust. The architecture used here is an outgrowth of work by Birrell, et. al. [Birrell86].

10 Delegation

When a user authenticates himself to a workstation, the user at the same time *delegates* to the workstation the right to speak on behalf of (act as a surrogate for) the user. This delegation is expressed in a certificate signed by the user's smart card at login. Delegation does not require any modification of ACLs. When the workstation accesses a remote service the workstation presents the delegation certificate to prove that the user authorized the surrogate. Note that remote access through a workstation does not require the remote system to reauthenticate the user. (The smart card does not play a role in any subsequent authentications or delegations.) Instead, the delegation certificate tells the remote system that the smart card trusts the

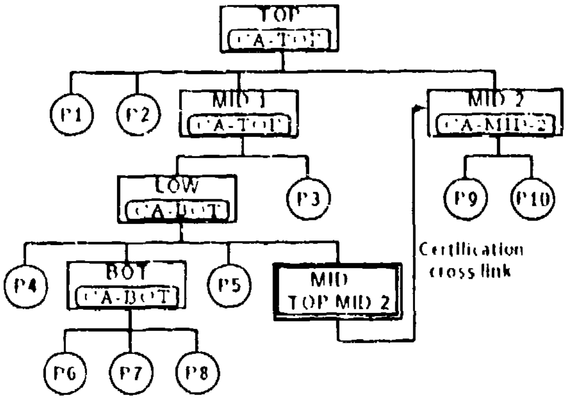


Figure 6: Symbolic link MID with certification cross link. CA BOT certifies CA MID 2

workstation to accurately reflect the user's commands. The remote system may wish to also authenticate the local workstation, however, using a challenge/response. Where there is a cascade of systems involved, each system delegates to the next system the right to act on its behalf (or the right to issue statements on behalf of the user), thereby propagating the ability to act as a surrogate for the original user.

Once the user delegates rights to a system, that system can act on the user's behalf even after the user logs out. To limit the damage in the case of a subsequent malfunction or compromise of a system, a properly functioning system terminates the delegation when it is no longer needed (e.g., at the end of a session) by destroying its copy of any secret key generated for purposes of that delegation and by notifying the parties with which they were communicating to no longer honor the delegation. (We assume users trust their systems while they are using them, but not necessarily after they logout.) As a backup, in case of system malfunction, delegations also time out, the timeout being set when the delegation is made. It is the responsibility of the system enforcing access to honor the timeout and delegation termination.

A delegation to a system implies the system may make any statements at all on behalf of the delegator. While restricted delegation, where the user specifies only a subset of statements such as a list of specific objects that may be referenced, seems desirable, the types of restrictions that might be useful are highly application-dependent and cannot be specified by a security architecture. Instead, we use the concept of user *roles* for such restrictions. A user authenticates himself using a DNS name that is the name of one of several possible roles, and these roles are represented as one-member groups in DNS, all containing the actual user name in their membership list. By delegating the rights of a specific role the user delegates rights to access only those objects that list the role on their ACLs.

11 Revocation

The architecture provides for a high degree of assurance that access is only granted when authorized. But once granted, revocation of access is not provided with the same degree of assurance. Although revocation is required and supported, the revocation may not take place in a guaranteed amount of time or before any specific event, and there is no absolute assurance that it will ever take place (except that there is usually some timeout or expiration that places an upper bound on the duration).

There are several things that one can imagine being revoked, all of which ultimately affect whether a principal has access to an object: access rights on ACLs, group membership, certificates for authentication, certificates for delegation, and authentication.

Immediate revocation is a difficult problem because it requires that either (1) systems not cache any information used to make access control decisions (public keys, group membership, ACL rights), or (2) there be a mechanism that reliably informs all systems using the access control information when a change has been made. Implementing (1) has an unacceptable effect on performance, and (2) is impractical since nobody can keep track of who is using the access control information.

Instead of immediate revocation, the architecture allows for "slow" revocation, where an application-by-application decision is made as to when, after a request to revoke, the revocation takes place. Most likely revocation will be determined by events: e.g., the next time a file is opened, the next time a user logs in, or when a delegation expires. Delayed revocation should be implemented in a way that causes users no surprises. Users maintaining ACLs, for example, might be informed that revocation has no effect on processes that currently have the file open.

A system is permitted to parse an ACL in advance, including expanding all groups named on an ACL, and to save that information for subsequent attempts by a principal to access the object. Removing a principal from a group or from an ACL will affect some subsequent access but is unlikely to affect accesses in progress. However, if (for example) the effect of this advance computation results in a user's access request being satisfied next time he logs in, even though he has since been removed from the group, then this implementation is not permissiole unless a way can be found to convince users that such behavior is reasonable.

Certificates used for authentication expire, but on occasion a certificate needs to be revoked in advance because a principal's private key has been compromised, or because the person changes affiliation and can no longer be trusted to access objects on whose ACLs he is listed. Certificates for authentication are stored in a few well-known places (most likely, in DNS), and all services that use certificates will look for them in these well-known places. Revoking a certificate means deleting each copy of the certificate from these places. This deletion is somewhat unreliable because DNS directories are replicated, but if DNS is functioning normally the changes will propagate to the copies in a reasonable amount of time. The certification structure in ISO's directory authentication framework [CCITT88b] also depends on the directory for the "security" of certificate revocation.

A system may cache a certificate (or the information in a certificate) but should periodically check the well-known places to determine whether the cache is still valid. Other techniques, such as checking the time a directory was last modified, can be used to make this process more efficient. A properly functioning system will not accept a certificate from any source other than a DNS server whom it trusts for revocation. In particular, the authentication dialog does not include transmittal of authentication certificates in place of those that should be obtained from DNS. In the event of compromise of a DNS server, or inability for a system to contact a server, revocation will not work.

Authentication cannot be revoked. Once a certificate has been used to authenticate a principal, that authentication is valid for as long as the original certificate was valid, or until the system chooses to stop using the authentication. Since authentication tends to happen at the beginnings of sessions when secure channels are created, authentication is not useful beyond the end of a typical session, and properly functioning applications that expect sessions to last for days or weeks should probably reauthenticate at intervals commensurate with the interval at which they check DNS directories for changes in certificates.

Like authentication, delegation times out but cannot be revoked once granted. However, delegation timeouts, tied to the lifetime of most sessions, will be far shorter than the certificate timeouts on which authentication depends. Both authentications and delegations are erased when no longer needed (at the ends of sessions).

Because delegation timeouts are relatively short, it is possible that a delegation will have to be renewed during a session before it times out. A facility is provided whereby such a renewal can be initiated by the first system in the delegation chain and propagated to other systems in the chain, provided that the user's smart card is still in place to sign a new certificate.

12 Mandatory access controls

The goal of the architecture is to provide mandatory (non-discretionary) access controls in all systems that implement discretionary access controls, but it is realized that some systems will never be used in a mandatory control environment and so implementation of mandatory controls is optional. Even if not enforcing mandatory controls, systems should be compatible with those that do.

DoD-style mandatory security as specified in the TCSEC is supported through labeling mechanisms controlled by the individual reference monitors. Every object and subject under direct control of a reference monitor has one or more *access class* labels, and mandatory access to local objects by local subjects is enforced in the usual manner.

A request originating from a remote system contains an access class label specified by the remote reference monitor, corresponding to the access class of the remote subject making the request. The local reference monitor uses this label, along with additional information about the remote reference monitor, to determine whether to allow the access. This additional information consists of certificates (obtained from DNS in a manner similar to the authentication certificates) that specify the *policy domain* and set of access classes for which the remote reference monitor is responsible. Access is granted only if the policy domain is appropriate (this domain may include information about the level of assurance of the remote system) and if the access class on the request is within the permitted set. The "cascading problem" discussed in the TNI [NCSC87] cannot be fully prevented except by system configuration, because none of the systems participating in the potential unauthorized write-down of information can be trusted to prevent it.

It is our intent to specify a commercial integrity architecture, perhaps based on the Clark and Wilson model [Wilson87], but work in that area remains to be done.

When both discretionary and mandatory access controls are applied to an access request, if either set of controls would disallow the request, then access is denied. In contrast to discretionary access controls, changes to mandatory access control attributes of principals and objects must take effect immediately. For example, security violations could occur if a request to "downgrade" or "upgrade" an access class does not immediately abort any accesses in progress that might no longer be allowed. The difficulty of implementing immediate revocation is mitigated by the fact that changes to mandatory attributes are rare, as noted above.

13 Problems not covered

The security architecture does not address all security concerns in computer systems. It concentrates on security problems that are unique to or exacerbated by distributed systems, such as authentication, secure communication, and global access control. Other problems in developing useful distributed systems, whether or not they have to do with security (such as global naming, synchronization, distributed databases, and assurance) are presumed to be addressed by other efforts, and a practical implementation of the security architecture may require solutions to problems in these other areas.

14 Status

The security architecture is intended for implementation across the entire Digital product line, including all operating systems, applications and hardware components. Any product acting on behalf of multiple users, or needing to take part in access control decisions, is affected by the architecture. When in place, the architecture will discourage the implementation of ad hoc, duplicative, and inconsistent security mechanisms in Digital software and hardware products. Of course, the security mechanisms will also be made available to customers for use by their own developers.

At this time of writing the details of the architecture (protocols, message formats, algorithms, etc.) are under development—little implementation has begun. Most of the groundwork and formal logic has been worked out, and functional specifications have been written.

References

[Ames83] Stanley R. Ames, Jr., Morrie Gasser, and Roger R. Schell, "Security Kernel Design and Implementation: An Introduction," *Computer*, Vol. 16, No. 7, July 1983.

[Birrell86] Andrew D. Birrell, Butler W. Lampson, Roger M. Needham, and Michael D. Schroeder, "A Global Authentication Service without Global Trust," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1986.

[CCITT88a] International Telegraph and Telephone Consultative Committee (CCITT), X.500, *The Directory - Overview of Concepts, Models and Services* (same as ISO 9594).

[CCITT88b] CCITT, X.509, *The Directory - Authentication Framework* (same as ISO 9594-8).

[DOD85] Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.

[ISO88b] International Standards Organization, ISO 7498-2, *Security Architecture.*

[NCSC87] National Computer Security Center, *Trusted Network Interpretation*, Ft. George G. Meade, MD, July 1987.

[Rivest78] R. L. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978.

[Wilson87] D D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1987.

GUIDELINES FOR SPECIFYING SECURITY GUARDS

William Neugent
The MITRE Corporation
7525 Colshire Drive
McLean VA 22102

Abstract: Security guards help to achieve trusted transfers across security boundaries. This paper summarizes guard policies, presents an overview of the trusted transfer process, and recommends guidelines for specifying security guards. Application and design considerations also are included. Key points of the paper are that (1) well-defined security policies and user requirements and a guard concept of operations are of fundamental importance, (2) the trusted transfer process includes functions performed by hosts or applications as well as functions performed by guards, (3) the guidelines should not be inflexibly applied to all guards, and (4) guards are not desirable solutions but are last resorts, to be used only when better solutions cannot be found.

1. Introduction*

A common requirement in both the Department of Defense (DOD) and the commercial world is the need to transfer data across security boundaries. Security guards help to achieve such transfers. With increasing needs for interoperability between systems, there are increasing requirements for security guards to control this interaction.

Several early attempts to develop high technology guards met with failure, in part due to lack of policy guidance on guards. In the past two years, new security policies have begun to address guards. This paper recommends more detailed guidelines to supplement the new policies.

At an internal computer security seminar of MITRE field sites, security guards were singled out as a topic of high importance and interest to military field organizations and a topic on which further guidance is needed. Subsequently, a second internal MITRE seminar was held in which participants in many guard acquisition efforts met to focus solely on guard requirements, approaches, and issues. The guidelines in this paper include insights gained from both seminars.

* This paper is derived from work performed under contract F19628-89-C-0001 for the United States Army, Europe (USAREUR) Office of the Deputy Chief of Staff for Operations, and from internal MITRE efforts to coordinate the guidance provided to several security guard development activities.

While these guidelines have no official standing, they have been used within MITRE and might serve as a basis from which to develop an official guard policy. The guidelines apply primarily to guards used between system high or dedicated mode systems, but can also be applied to guards used between the different levels within a multilevel system.[1]

2. The Requirement

The generic requirement to be satisfied is the trusted transfer of data across security boundaries. A security boundary exists between two systems when the systems operate at different security levels, e.g., Top Secret and Secret. Communication between two such systems involves the transfer of data at a classification level subsumed by both systems. For example, a Top Secret system would only be able to send data classified Secret or below to a Secret system.

The general security objectives in communicating across security boundaries are to prevent leakage and penetration. The primary concern normally is to defend against unauthorized disclosure of "high-side" (e.g., Top Secret) data to "low-side" (e.g., Secret) users. This might be caused by high-side errors, by malicious high-side software, or by active penetration from the low side. Other concerns include defending against modification or destruction of high-side data as well as denial of service to high-side users, both caused by actions originating on the low side (e.g., worms, viruses).

Ideally, the communicating systems should be able to defend themselves against these exposures. Unfortunately, most systems are not considered sufficiently trustworthy to do so. Most military automation systems operate in dedicated or system high mode, in which the system is not trusted to segregate work being done at different security levels.* In such operation, all system users must have security clearances for the most highly classified data on the system, and all output is protected as though it contains the most highly classified and most restrictively controlled data processed by the system, until the output is reliably reviewed and its actual classification and sensitivity verified. Policy disallows communication--without reliable security control--between systems operating at different security levels (except that some policies allow one-way, receive-only links from low to high systems, e.g., wire service links). Furthermore, without a reliable review mechanism, magnetic media removed from the system must remain classified at the level of the system.

* Although the term "system high" is officially defined in policy documents to be a distinct operating mode, the term normally is used in the field to encompass both the dedicated and system high modes.

Guards satisfy this trusted transfer requirement. Furthermore, the requirement for guards will not disappear as more trustworthy systems become available (e.g., systems that satisfy requirements for classes B1 or higher in the DOD Trusted Computer System Evaluation Criteria, hereafter referred to as the Orange Book)[2]. The majority of systems still will continue to operate in dedicated or system high mode (e.g., using class C2 or less trustworthy technology). Even systems that operate in multilevel mode will need guard functions to transfer data between system high objects at different security levels (where typical objects include files, messages, and reports that were either received from system high systems or created during system high work sessions). The nature of guards will change, but the need for guards will not.

3. Past Difficulties

While some guards have been successfully developed and used, others have not fared so well. At least four major efforts to produce guards for the military have failed, in the sense that the guards were not used operationally[3]. Several guards that are being used are used only with great reluctance, due to their extreme awkwardness or the heavy burden they place on operation. Still other guards are accepted only grudgingly: users believe that the benefits justify the costs, but resent the costs nonetheless. Several guards were used operationally and then deactivated, either because they failed to prevent the disclosure of sensitive data or because they prevented the flow of data that should have been released. Finally, some guards are being used carelessly and threaten to compromise the very data that the guards were installed to protect.

On the surface, there are many explanations for these difficulties:

- Some guards sought high degrees of technical security (e.g., class A1), and were as a result complex, expensive, and time-consuming to develop, and once developed were inflexible and difficult to change.
- Some guards introduced additional workstations, hosts, or specialized hardware as well as complex software.
- Some guards required substantial hardware or software changes in the systems being supported.
- Some guards imposed cumbersome operation and administration.

One underlying reason for these difficulties was lack of sufficient policy guidance. Because of this lack, some guard efforts attempted too much, others attempted too little, and others were misguided. In the past two years, however, a number of new security policies have addressed guards. The following section summarizes two of the new policies and also surfaces a policy issue.

4. Current Policy

Before summarizing new guard policies, it is important to raise a fundamental policy issue. Many policies exist that tell how to downgrade, sanitize, or decompartment particular types of data or that specify classification requirements for data on a particular program. Sometimes these policies are simple and clear, but often they are complex, ambiguous, inconsistent, or unavailable. It is not possible to develop an effective guard without clear, thorough policy on classification, downgrading, sanitization, decompartmentation, and releasability. This issue cannot be resolved in this paper, but must be addressed in any guard application.

The most widely applicable new policy impacting guards is that in DOD Directive 5200.28[1]. DOD Directive 5200.28 states that, if a system is not at least class B1, then downgrade of output from that system requires manual review by an authorized person. The implication is that, without the level of trust implicit in a class B1 or higher foundation, it is not permissible to rely on software to downgrade data. In a statement applicable only to intelligence systems, DOD Directive 5200.28 expands on this general policy by saying that fully-automated software downgrading is allowed (1) if the involved system is at least class B1, (2) if the output remains classified (though at a lower level than the originating system), and (3) if the downgrade capability is approved in the accreditation for that system.

While this policy is helpful, for the most part it does not explicitly address guards and thus leaves most guard questions unanswered. For example, there is no guidance in the DOD Directive on what is required in performing human or automated review. There is no guidance on what assurances are needed in a guard that is separate and independent from the system it is supporting and that resides on a processor dedicated to the guard function.

A policy that addresses guards more explicitly is the National Telecommunications and Information Systems Security Advisory Memorandum (NTISSAM) on Office Automation Security[4]. The policy states that copying data to a medium classified at a lower level than the system is an "extremely dangerous practice" and that procedures established by the Information System Security Officer (ISSO) should be followed. The policy further states that, in establishing and using the procedures, responsible people must consider and accept the risks. Appropriate procedures "in some instances" are as follows:

- Format a new (i.e., never used) medium.
- Copy the data to the medium.
- Carefully examine the medium; check that no other data has been copied; if feasible, print out the entire medium.

This policy is especially helpful and encompasses a large number of systems in that it applies not only to standalone personal computer systems but also to terminals connected to mainframes and to workstations in a local area network (LAN). Unfortunately, it applies only to the downgrade of data onto a physical medium.

Other new policies on guards also exist. Some are classified and some apply only to particular agencies within the DOD. But people responsible for defining guard requirements and specifying design approaches still have been left much opportunity for error. The remainder of this paper supplements the above policies by clarifying the nature of guards and recommending guidelines for specifying guards.

5. Functional Overview

One difficulty in discussing guards is that the term is used in many different ways. For example, whereas some "guards" support manual review and downgrade of data, other "guards" do little more than validate a Cyclic Redundancy Check (CRC). One view is that a "guard" performs *only* a checking function, not the actual downgrading, sanitization, or decompartmentation. Rather than choose a narrow definition of guards that is inconsistent with some of these common usages, this paper uses a broad definition:

> *A guard is a process (or set of controls) that helps to control trusted transfers across security boundaries*.

The significant feature of this definition is that it places guards into the broader context of trusted transfers. While guards serve a variety of roles in helping to achieve trusted transfers, the trusted transfer process itself is more fundamental and more complete. Indeed, according to some views of what a "guard" is, many trusted transfers do not even require guards. For example, there are accredited systems without "guards" in which system high hosts are trusted to produce and verify output at less than system high, where the data is well-structured and where no changes are made to the data or its labels.

So, *although this paper uses the term "guard," the scope of the paper encompasses the trusted transfer process as a whole; some of the functions discussed thus will be performed by hosts or applications rather than by guards*. Figure 1 is a functional overview of the trusted transfer process.

The functions are grouped into two layers: application and communication. The emphasis within application layer functions is on *examining* the data whereas the emphasis within communication layer functions is on *moving* the data. The term "trusted" is not used in an absolute sense, but has different meanings for different systems (e.g., normally the low side is untrusted relative to the high side).

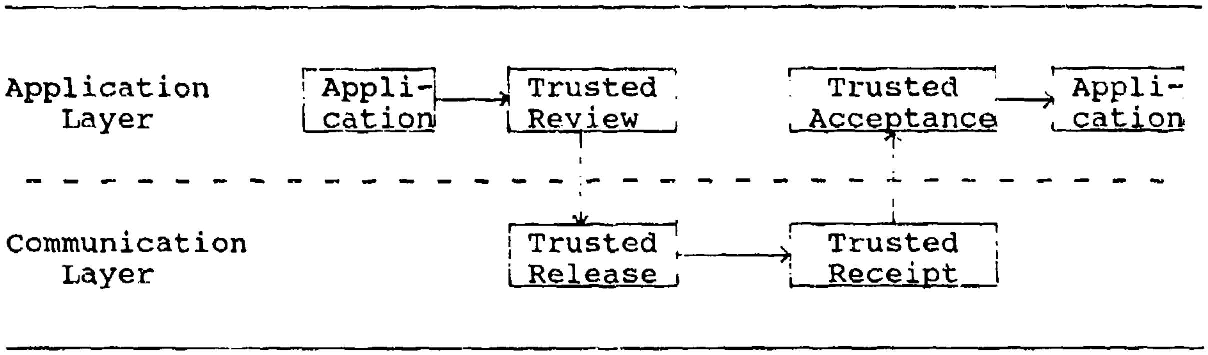


Figure 1. The Trusted Transfer Process.

The most significant trusted transfer functions are those residing within the application layer: trusted review and trusted acceptance. The trusted review function verifies the actual classification of the data or downgrades, sanitizes, or decompartments the data to lower the actual classification. The trusted review function might also remove control or release markings and caveats. The final result of this verification or transformation is to reduce the data classification from the overall classification of the system or single-level object in which the data resides. Subsequently, the trusted review function officially authorizes release of the data to the low side. Note that the topic of changing data classification or sensitivity is complex and is further discussed in sections 6.1.1 and 6.2.10.

The trusted acceptance function protects the receiving system or application from penetration. Trusted review normally is associated with high-to-low transfers and trusted acceptance with low-to-high transfers, but in actuality both functions are applicable regardless of the direction of data flow.

The communication layer functions are trusted release and trusted receipt. Taken together, the trusted release and receipt functions are responsible for transferring the data between the review and acceptance functions, while preventing unauthorized data leakage from one system to the other and ensuring communication integrity (e.g., authentication; protection from modification, insertion, deletion, and playback)[5]. The trusted review function should insert an integrity-check (e.g., a CRC) that is checked by the trusted release function to ensure that the data has not been changed subsequent to output from the trusted review function.

Few guards perform all of these trusted transfer functions. Most focus on trusted review, e.g., the Message Flow Modulator (MFM), the National Aeronautics and Space Administration (NASA) Restricted Access Processor (RAP). Some guards provide only trusted receipt, in the form of one-way communication paths, e.g., wire service links and media transfers (both of which often do not require an explicit "guard") and links in which a guard provides protocol mediation to ensure one-way communication. One interesting example of a one-way path is an operational military system in which (1) the low-side system writes to a disk, (2) a switch is thrown that prevents further low-side disk access, and (3) the high-side system reads the disk (but cannot write to it).

Trusted acceptance usually is not explicitly identified as a guard function and is entrusted to the receiving host. Where risks are minimal, as with wire service links, little attention to trusted acceptance is needed. Lately, however, increasing occurrences of worms and viruses make it apparent that more attention must be placed on the trusted acceptance function. Furthermore, some guards (e.g., the United States Army Forces Command Security Monitor) have used extensive filtering and format checking of low-side input to prevent penetration of the high-side system.

One function not shown in figure 1 is administration of the trusted transfer process. Software needs to be loaded and maintained. Transfers need to be audited. Errors and problems need to be brought to the attention of responsible people. Such administrative activities are an important part of the process and must be carefully planned.

Wherever there is to be a trusted transfer, responsibility must be assigned for all of the trusted transfer functions, even though the particular function might be trivial for a given application. Often some or most of these responsibilities will be assigned to hosts or applications rather than to guards as such. Note that, even though a guard is being added, where hosts or applications must fulfill some of the trusted transfer responsibilities, the hosts or applications might have to be strengthened (or at least tested more thoroughly) in order to achieve sufficient security.

6. Guidelines

This section presents guidelines for specifying security guards. Although the term "guard" is used, the guidelines apply to all components of the trusted transfer process, regardless of whether they are encompassed within something explicitly referred to as a guard. The guidelines are intended to apply to guards in which there is an electrical connection between the high and low systems rather than to guards in which media transfer is used. Nevertheless, many of the features and assurances can and should be applied to media transfers.

These guidelines interpret and supplement the Orange[5] Book and in fact demonstrate the versatility of the Orange Book[2]. While the Orange Book was not developed to address guards, much of its contents are applicable.

Because so many different types of guards are possible, an important assumption of these guidelines is that they not be monolithically or inflexibly applied to all cases. Were these guidelines to be transformed into official policy, users should be able to submit written justification for exemption from specific policy statements.

Another important assumption of these guidelines is that it is not necessary for the guard itself to be a multilevel secure (MLS) system in the pure sense of the Orange Book (i.e., a class B1 or higher system, with all of the B1 features and assurances). The reason is that MLS features and assurances often just are not applicable to what many guards do. That is, guards basically implement a single trusted function and do not support direct users and the sharing of data as general purpose systems do. In addition, sometimes the guard's trust derives not from MLS assurances but from the guard's independence and from configuration management of the guard hardware and software. Of course, the more trustworthy foundation provided by an MLS system is an excellent base upon which to build guard functions, but until MLS systems are comfortably within the state of the art, they can be a costly, risky foundation.

Although the guidelines do not require guards to have a full MLS foundation, the argument still might be made that the guidelines ask too much, in light of existing operational guard precedents. In some cases this may be true, but one purpose of the guidelines is to chart a course towards continued improvement.

The remainder of this section expands upon the features and assurances in the Orange Book. Note that the features might be implemented in a distributed fashion, with different features or even portions of one feature implemented in different computers.

6.1 Features

6.1.1 Trusted Review

Trusted review requirements differ for human and automated review. The following guidelines distinguish the two cases and also include requirements that apply regardless of whether human or automated review is used. Automated review is appropriate only if data releasability can be reliably determined based on data structure and content. That is, review criteria must be sufficiently predictable to be automated. Where this is not the case, human review is required. In any case, trusted review is only practical if policy exists that explicitly establishes the rules for downgrading, sanitization, decompartmentation, and classification. The need for such policy is fundamental. Sometimes automated review can be made feasible by working with policy authorities to revise and simplify applicable security policies.

Where human review is used:

o The system supporting the human review function shall be at least a class C2 system. This is necessary to obtain proper authentication, accountability, and so forth for the human reviewer. This requirement applies whether human review is performed in the guard or is embedded within the system being supported.

- Reviewers shall be fully cleared and authorized for all data that might be received by the guard. Encompassed within the term "reviewers" should be two roles: "data preparers," who prepare, assemble, or initially review the data, and "release authorities," who perform the final review and authorize release of the data. Normally there will be many data preparers and only a few release authorities. In all cases where human review is used, the authority to release data shall be explicitly and officially assigned. If the trusted review function includes data transformation (e.g., removing or changing data to lower the classification or sensitivity), then the data preparer and release authority roles <u>shall</u> be performed by different people.

- Reviewers shall be qualified to recognize data that can be released and shall be familiar with the most sensitive types of data that cannot be released. To be "qualified to recognize data" means to be very familiar with data content, not just data structure. Where feasible, the reviewer should be the data owner of the data to be released. Sometimes it is unrealistic to expect the reviewer to intimately know all data that <u>cannot</u> be released; in a shared system, that other data might not be part of the reviewer's day-to-day work. Therefore, the reviewer must not release any data with which he is not familiar. The reviewer must be very familiar with applicable classification guidance and releasability policy.

- Reviewers shall be trained to examine data content in addition to data labels and shall be trained not to place total reliance on data labels, especially where the labels appear to understate data classification or sensitivity. The reviewer shall not override or second guess labels affixed by data originators, but shall contact the originator where clarification or confirmation is needed. A problem commonly encountered by reviewers is that data originators often do not indicate releasability. This can be a major problem when originators are the only people authorized to determine releasability. Where feasible, originators must be required to indicate releasability when data is created.

- All data (including system control data) to be released shall be accessible to the reviewer. This does not mean that the person must review all of the data, but that he could if he so chose. Note that the human might review text and an automated process review control (e.g., protocol) data. Where large amounts of data are involved, the human should not be forced to review all of the data. The human should, however, be forced by the guard (or at least instructed by procedural guidelines) to review the beginning and end of any file or data stream, to ensure that the correct data is being transferred and that no "straggler" data is attached. Human judgment should be relied upon to choose which intermediate data samples to review (e.g., to check for "interlaced" data), but the human should be forced (or at least instructed by procedural guidelines) to review at least a minimum percentage of the data. One DOD policy requires that the review encompass "a random sample comprising not less than ten percent of all media storage locations (including beyond end-of-file mark)"[6].

Where automated review is used:

- If the automated review function is embedded within the system being serviced, the system shall be at least a class B1 system.

- There shall be a reliable means to determine releasability. Candidate techniques include the following:

  - Where releasability can be determined by the presence in text of specific words, a text scan can be used, either of the full text or of predetermined fields. Careful planning is needed, however. One operational guard used a text scan to filter out words that <u>were not</u> allowed to pass. Unfortunately, sensitive information managed to flow past the guard's checks and the guard was deactivated. Another operational guard used a text scan to identify words that <u>were</u> allowed to pass. Unfortunately, much data that should have been allowed to pass was filtered out. This guard also was deactivated.

  - For highly-structured messages or for database output, the review can validate the format of every field and the relationships among fields, as well as the value of selected, predetermined fields. Where labels are present, they should be checked, but releasability should not be totally dependent upon this one check.

- Review functions and data releasability criteria must be approved by appropriate authorities (e.g., the originators or owners of the data at risk) and must comply with applicable policies.

Regardless of whether human or automated review is used:

- There shall be some independent means to validate review actions, so as to pose the risk of detection to someone subverting the process. For human review, this validation might be an automated review (e.g., to search for unauthorized security labels). For automated review, validation might be a human check. For example, there might be an optional human review role that can be activated or deactivated as required, e.g., the human review role could be filled during initial operation, during high-risk periods such as military exercises, and during periods in which there are changes to or problems with automated review.

- There shall be a capability to audit both the change in classification or sensitivity of an object and the transmission of that object to another system. It should not be necessary for both the guard and the high-side system to perform the audit--one record normally is sufficient.

- All data being reviewed must be properly handled, e.g., no extraneous data inserted, data sequence maintained.

- If the threat warrants, there shall be some means to ensure that reviewed data is not supporting covert channels (above a certain bandwidth). Candidate techniques include overwriting unused communication protocol fields and displaying nonprintable characters by using a unique displayable equivalent for each nonprintable character.

6.1.2 Trusted Release

- Data shall be released only if the release has been approved by the review process; if there is more than one review process, reliable identification of the review process shall be ensured.

- Communication integrity shall be ensured (e.g., authentication; protection from modification, insertion, deletion, and playback)[5]. In some systems, untrusted components process the data after its output from the trusted review process and before its input to the trusted release process. To ensure data integrity, an integrity check (e.g., CRC) should be added to the review process and checked by the release process. An additional check used by some guards is to check the classification label that was set by the trusted review process and "locked" by the integrity check. Finally, the trusted release modules of some guards perform still additional checks, e.g., to verify that the releasing individual is authorized and that the destination is authorized.

- Some guards have strengthened the integrity check (see above bullet) by encrypting the CRC; others have not. The need for encryption to further protect the integrity lock should be determined based on a vulnerability analysis and on guidance provided by the accreditor(s). Note that CRC encryption is an effective defense against system errors, but not against attacks by malicious software.

6.1.3 Trusted Receipt

- There shall be some means to prevent data leakage (from the high system) during receipt. This might be achieved via a pure receive-only circuit rather than by adding software checks or by modifying communication protocols to effectively achieve one-way transfer.

- Communication integrity shall be ensured.

6.1.4 Trusted Acceptance

- Allowable data flow shall be restricted in such a way as to prevent penetration and to prevent infection by worms and viruses. Note that some of this protection is available from discretionary access controls, virus detectors, and other mechanisms used to control access within system high systems.

  - Executable software shall not be transferred from a low to a high system.

  - Usage of a high-side system by low-side users shall not be allowed. Low-side users shall be allowed to forward only data (not commands or queries) to a high-side system.

- If the threat warrants, data content and structure can be checked (1) for correctness and (2) to ensure that there is no extraneous data.

6.2 Assurances

Many of the assurances in this section are at the class B3 level of the Orange Book due to the increased configuration management, security testing, covert channel analysis, trusted facility management, and trusted recovery at the B3 level. Differences and omissions from Orange Book assurances reflect differences between guards and general purpose systems. The main differences are in the areas of system architecture, formal policy models, developer clearances, and documentation. Where the Orange Book is referenced without change, Orange Book terms must be interpreted to reflect the trusted transfer process rather than a general purpose system. Section 7 discusses how these assurances might vary as the classification differential of supported systems varies.

It must be emphasized that satisfying these requirements should be much simpler than building a class B3 system. First of all, a guard is a much simpler object than a general purpose system and need not include class B3 features such as internal labeling and mandatory access control based on labels. Secondly, guards are not required to meet the difficult class B3 system architecture requirements. Guard assurances normally are based more on multiple independent checks than on single trusted checks. The primary reason class B3 assurances are needed is to ensure effective testing, management, and operation of the guard.

6.2.1 System Architecture

- The guard shall be protected from external interference or tampering (e.g., by modification of its code or data structures). Possible approaches include physical separation of the guard (on a separate system or board), use of a software integrity check, and use of MLS-based separation. Guard modules shall be designed such that the principle of least privilege is enforced. Note that use of the guard for a physically separate, independent check of a function serves to increase the level of trust associated with that function, since two independent checks are more trustworthy than a single check. This concept has a solid basis in precedent in that it is a variation of the idea underlying two-man control and separation of duties.

- It is an objective to satisfy as many of the class B3 architecture requirements as feasible and applicable. Nevertheless, different trusted transfer components often reside in different systems and have different architectural requirements associated with them. Furthermore, the use of physically separate, independent checks (see above bullet) often is more practical for guards than the use of pure class B3 approaches, in which individual functions are trustworthy unto themselves.

6.2.2 System Integrity

- See Orange Book (class B3).

6.2.3 Covert Channel Analysis

- See Orange Book (class B3). The covert channel analysis shall be supplemented with an analysis of the difficulty of implanting malicious software to exploit the channels. Where the classification differential between the supported systems is small (and the security risks accordingly low), this analysis can supplant the standard covert channel analysis. Where covert channels are a significant vulnerability, symptoms associated with their use shall be identified and included in the Trusted Facility Manual.

- o Covert channel analysis activities normally should require hours or days rather than months of effort, since the likelihood of a covert channel being exploited is very low. Channel defenses should not be allowed to have an inordinate influence on guard design and operation.

6.2.4 Trusted Facility Management

- o See Orange Book (class B3).

6.2.5 Trusted Recovery

- o See Orange Book (class B3).

6.2.6 Security Testing

- o See Orange Book (class B3).

6.2.7 Design Specification and Verification

- o See Orange Book (class B3). An exception from the Orange Book is that the specific guard approach determines whether there is a requirement for a formal security policy model. (Where there is no requirement for a model, there also is no requirement for (1) a formal proof that the model is consistent with its axioms or (2) a convincing argument that the descriptive top-level specification is consistent with the model.) For example, a formal model might not be needed in cases where trust is dependent on physically separate, independent checks rather than on individual "trusted" checks, especially where the checks reside on commercial software that is of class C2-level trustworthiness. That is, in these cases the benefits of a model probably would not justify the costs. Cases where a formal model would be of particular benefit include cases in which a class B2 or higher component is used to provide trusted separation in support of trusted release and receipt.

6.2.8 Developer Clearances

- o Software specific to a particular guard application shall be developed by people cleared to the level of the most highly classified and most sensitive data that might be processed by the guard. Note that this allows uncleared people to develop a generic guard and also allows generic guards to be used without modification, but requires that adaptation for specific uses be done by cleared people.

6.2.9 Configuration Management

- o See Orange Book (class B3).

6.2.10 Documentation

- User Requirements and Data Flow Analysis. Requirements analysts, working with data owners and security classification experts, shall perform a thorough analysis of data flow to determine whether the systems involved can be made to operate at the same security level, thereby alleviating the need for a guard. This normally is a preferable approach to use of a guard approach. If the need for a guard cannot be alleviated, analysis shall be performed to identify data types, classifications, formats, throughput, and response times, as well as applicable classification guidelines and downgrade, sanitization, and decompartmentation policies. The analysis also shall identify:

    - All data that aggregates to higher classification levels or greater sensitivity. On one guard effort, careful aggregation analysis quadrupled the amount of data that could not be released to the low side.

    - All data that might not be reliably classified (e.g., due to difficulty in determining the proper classification to assign).

    - All data whose classification cannot be determined on inspection (e.g., telemetry data whose classification depends on its source, not on data content).

    - The impact of classification guidelines or releasability policy being changed, e.g., due to changing technology, changing international affairs, or tactical command decisions made during crises. Some guards have included the capability to bypass the guard (e.g., in a wartime crisis). This capability should be very tightly controlled.

    - Requirements for changing the security labels and associated markings relating to categories, handling, classification authorities, ownership, control channels, and declassification statements.

    - Whether "cascading" might occur. The "cascading problem" is identified in the Trusted Network Interpretation of the Orange Book[5]. As applied to guards, cascading occurs when data flows from a high to a low system and then to a still lower system. Where cascading occurs, the high-side accreditor(s) must be aware of the threat and be satisfied that low-side defenses are adequate.

- Risk Assessment. This is needed to determine whether the guard provides adequate protection. The document shall list all threats and identify how the guard defends against them.

- Concept of Operations. This is a high-level document that shall be prepared and approved before guard development is begun. This is needed to show not only what the guard will do and how, but also what procedures and support will be needed for guard operation, administration, and maintenance. This is the key document in obtaining the understanding and commitment of involved people. The document is the basis upon which involved people decide that the benefits of the trusted transfer outweigh associated development and operational costs and that costs are justified in light of the risks. This document must be acceptable to accreditors, data owners, data users, system managers, security managers, program managers, and offices responsible for hard copy transfers between the involved organizations.

- Memorandum of Agreement. This shall address accreditation requirements of supported systems. The document should include a description and classification of the data, the clearance levels of the users, a designation of the accreditor who shall resolve conflicts among involved accreditors, and a brief description of the guard. This document is not required if all systems have the same accreditor(s). (See DOD Directive 5200.28, paragraph D.8.1)

- Security Features User's Guide. See Orange Book (class B3).

- Trusted Facility Manual. See Orange Book (class B3).

- Test Documentation. See Orange Book (class B3).

- Design Documentation. See Orange Book (class B3). The exception from the Orange Book is that the specific guard approach taken determines whether there is a requirement for a formal description of the security policy model (and for other requirements deriving from the model).

7. Application in Different Environments

While the trusted transfer process is constant when viewed in generic functional terms, in practice there are many different ways in which trusted transfer is accomplished. Some of the differences derive from different functions being performed and impact the selection of guard features. These differences can vary widely and are discussed no further in this paper except to reaffirm the statement that the guidelines must not be monolithically or inflexibly applied to all cases.

Other differences among guards derive from the varying classification differentials of supported systems. For example, one guard might support data downgrade from a sensitive but unclassified system to an unclassified system, whereas another guard supports data downgrade from an intelligence system to an unclassified system. These differences impact primarily on guard assurances, and are discussed in this section.

Many of the assurances included in section 6 are at the class B3 level. According to DOD Directive 5200.28 (enclosure 4), class B3 systems can be used in environments where the "risk index" is three[1]. (The term "risk index" is not strictly applicable to guards, but is used here somewhat loosely to quantify classification differentials.) Risk index is defined as the difference between the minimum user clearance and the maximum data classification. Examples of systems with a risk index of three are systems with uncleared users and Secret data and systems with Secret-cleared users and intelligence data.

This is not meant to imply that the generic guard specified in this paper is equivalent to a class B3 system, because it's not. The DOD Directive simply provides a starting point for determining how much trust to place in a guard that includes many class B3 assurances. An additional factor to be considered is that many guards, having been developed (or adapted) by cleared people and being physically independent from supported systems, are therefore better protected against malicious software and thus have some of the characteristics of a "closed security environment"[7]. Such guards should be sufficiently trustworthy to support systems whose classification levels are more disparate than could normally be supported by class B3 technology. Another reason guards can support greater classification differentials than the risk indices might imply is that guards typically support only data flow between systems, not full-capability usage of one system by users from another system.

Where the differential in classification levels of the supported systems is smaller (e.g., risk index equal to one or two), class B3 assurances might not be needed. Even where the classification differential is small, however, care must be taken to retain sufficient system architecture assurance, security testing, configuration management, and documentation, since those are key to guard trustworthiness. Note that in all cases the specific guard mechanisms used must be approved by the responsible accreditor(s).

A final environmental consideration regarding classification differential is that some applications simply are too sensitive to risk errors that result in, for example, disclosure of highly classified data to uncleared people. So some guards should be entrusted to downgrade only to the Secret or Confidential levels. (DOD Directive 5200.28 suggests this for fully-automated guards used with intelligence systems[1].)

Classification differentiºl is not the only criterion affecting risk. Another important criterion (especially where human review is involved) is the amount of data that must be transferred: the greater the volume, the greater the risk. Where the volume is high, .uch effort should be spent to change the operating levels of the systems so that a guard is not needed. Where the volume is low, greater classification differentials can be supported.

8. Design Considerations

It would be a serious mistake if designers interpret this paper to imply that guards are desirable or are simple to achieve and thus casually insert guards into architectural plans. The fact remains that guards are a last resort. The goal is to formulate an architecture in which guards are not needed.

One decision that must be made in designing a guard is whether it should be external to or embedded within the system being supported. The advantage of an embedded guard is that it avoids the addition of an extra component that might add to the management burden and represent a single point of failure. Advantages of external guards (over embedded guards) are that they reduce the risks of (1) corruption by the more complex, less trustworthy host software, (2) penetration by users, and (3) communication headers being used for covert channels (since the guard can create or at least review the headers).

Based on lessons learned from past successes and failures, following are desired guard design features:

- Do not require a full-time human reviewer.
- Allow controlled release (1) of any type of data, (2) to any compatible system, (3) at any classification level (where authorized).
- Avoid the use of additional hardware, except that which can be housed (e.g., as a board) within existing components.
- Do not require changes to commercial software in systems ŀeing supported.
- Employ guard-specific software tha⁺ can be developed quickly and with minimal cost and risk.
- Require minimal procedures to operate, maintain, and administer.
- Avoid creation of a single point of failure or a communication bottleneck.
- In distributed processing environments, collocate guards with the data owners of data to be released.

9. Conclusion

Experiences with failed guards and with guards operating at high cost or risk affirm the need for additional guidance. The operational requirements for trusted transfers are too pervasive and critical to ignore lessons learned from past difficulties.

This paper codifies into guidelines many of the lessons learned. Discussion and guidance in this paper do not answer all questions, but represent a first step. Application of these guidelines should improve the effectiveness of trusted transfers and reduce the likelihood of failure in guard development efforts. As these guidelines are refined with use, they might provide a basis from which to develop more detailed official policy.

Acknowledgments

The author is indebted to Mr. Len Busic of the Defense Intelligence Agency for his comments and insights. The author also is grateful for suggestions from Diana Akers, David L. Baldauf, Maureen H. Cheheyl, Wade R. Gerhart, Mark J. Goldstein, H. Craig McKee, Brian W. McKenney, Samuel I. Schaen, Peter S. Tasker, and John M. Vasak, all of MITRE. Thanks also go to conference reviewer number 72.

References

[1] DOD Directive 5200.28, "Security Requirements for Automated Information Systems (AISs)," Deputy Secretary of Defense, March 1988.

[2] DOD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria, Deputy Secretary of Defense, December 1985.

[3] Neugent, W., "Security Guards: Issues and Approaches," IEEE Communications Magazine, Vol. 26, No. 8, August 1988.

[4] NTISSAM COMPUSEC/1-87, Advisory Memorandum on Office Automation Security Guideline, National Telecommunications and Information Systems Security Committee, National Security Agency, January 1987.

[5] NCSC-TG-005, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, National Computer Security Center, July 1987.

[6] JCS Pub 6-03.7, Security Policy for the Worldwide Military Command and Control System (WWMCCS) Intercomputer Network, The Joint Chiefs of Staff, April 1988.

[7] CSC-STD-003-85, Computer Security Requirements--Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, National Computer Security Center, June 1985.

SECURITY FOR EMBEDDED TACTICAL SYSTEMS

Howard L. Johnson
Information Intelligence Sciences, Inc.
15694 E. Chenango, Aurora, CO 80015

Chuck Arvin
CTA INCORPORATED
7150 Campus Drive, Suite 100, Colorado Springs, CO 80920

Abstract

We take an embedded system to be a computer that is a component of a larger system comprised of other electromechanical components. It may be stationary. If deployable, it can be in garrison (home environment), in storage, in shipment, deployed, in maintenance, or captured (in the hands of the enemy). Each embedded system has its own unique characteristics, but those that might be present and are important from a security standpoint are sensitivity, criticality (integrity and service assurance), complexity (different component policies or build times), production in multiple units with wide distribution, partial or total autonomy, operation in an unfriendly or hostile environment (a computer that is part of a tactical DoD system) and response driven operations (such as real time). This paper addresses security for embedded systems.

Introduction

The computer has served us for several decades as a monolithic system and is now called upon to be integrated into other systems. An important example is a network controller. Soon the computer will become part of most vehicle systems, play a role in vital control functions, and be a principal interface with the human through voice and image interpretation as well as image and audio response. Many autonomous and semiautonomous electromechanical devices will house one or several processors to interpret sensors, pilot, plan, control, map, navigate, and control end effector (e.g., gunner) functions.

The security community has had great success in protecting information confidentiality. The word "security" has a broader meaning including guarding against danger, making functionality certain, and rendering loss or failure impossible. There has been significant work on ensuring integrity and assured service, but uniform guidance is lacking. As Figure 1 suggests, the science of security protection will continue to expand. Current modeling, specification, and assurance in the presence of intelligent hostile threat are applicable to the expanding protection problem.

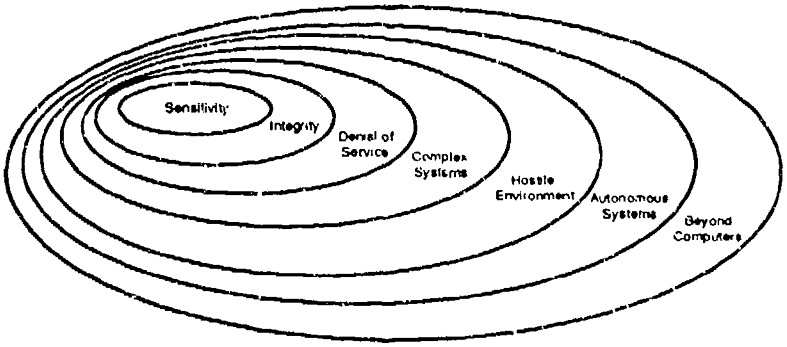


Figure 1. Expanding Security

An Embedded System

An embedded component is conceptually similar to a network component (Figure 2). Manufacturing protocols (MAP) continue to evolve to define the communications interfaces between standard components. One embedded computer may cooperatively act with several other embedded computers as in a military aircraft. The protection of data during communications is difficult in networks since network connections must operate in a hostile environment. In comparison, the system containing an embedded component may operate in an environment even more threatening than the network.

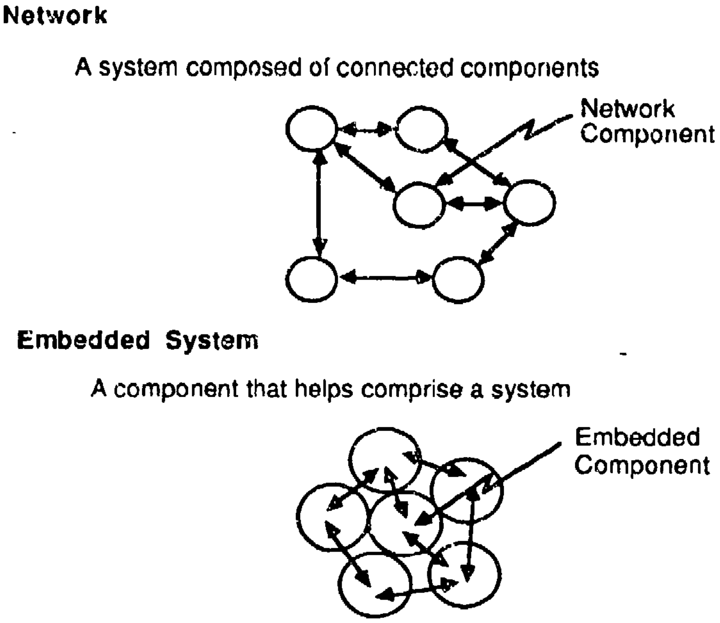

Network
A system composed of connected components
Network Component
Embedded System
A component that helps comprise a system
Embedded Component

Figure 2. Complex System

Historical Treatment of Security in Tactical Systems

Tactical military systems and even civilian transportation systems have not traditionally considered data system security. From a historical perspective, only sensitivity threat was considered important. In the war zone, most classified information was tactical and highly perishable. In a fast moving situation there may be no time for the enemy to exploit it. Exchanging battle plans and orders depended on communications security (COMSEC) and encryption prevented exploitation. Everyone in a Tactical Operations Center was cleared to the highest level and was assumed to have total need-to-know. Logging on and passwords were felt to be unnecessary. Explosive destruction or capture were more likely and effective than a data system security attack. Spending funds on defense of the latter was not warranted.

Threat of a Data System Attack

There has been recent growth in the use of computers in all types of systems performing complex and vital functions. We see the development of functional and higher level application specific languages that will allow "programming" capability in real-time with much the same opportunity for a malicious threat as exists today with standard languages. The real and present opportunity exists to divert or nullify a weapon system with a data system attack. The potential exists for sophisticated terrorist attacks that use no explosives and hold no hostages, except the computers themselves.

Read only memory (ROM) is not impenetrable and is easily changed by maintenance personnel. Random access memory (RAM) is used in command and control systems for data acquisition and processing, and depending on the computer, may be loaded with executable code. Dependence on sensor systems and the ability to command remote systems affords many opportunities for spoofing, jamming, confusion, overload and agility as potential attacks. A real time battle planning and management function may well be the single most important capability existing in a tactical environment. In the near future it may be embedded as part of a command post capability. Implementation may be in software because of configuration management and control required for such a large piece of code.

Because of modern day weapon system capabilities, the stakes are high and the role of spoiler (destroying the capability) more lucrative. There are many opportunities for attack (Figure 3) not only during development but during deployment and maintenance. We must begin to solve the problem effectively now, so that the threat may never mature.

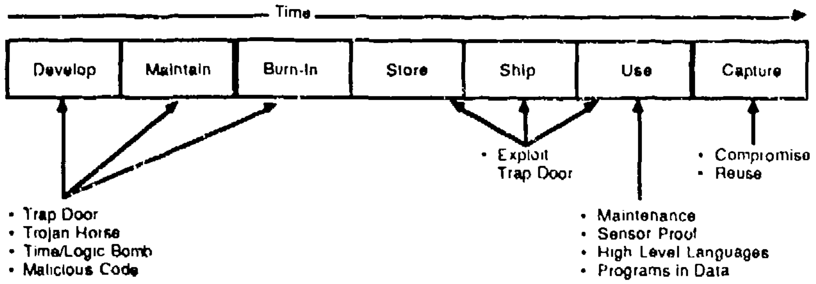


Figure 3. Life Cycle

Sensitivity

The Orange Book [1] stands as the preeminent basis for formal security policy. It has gained intellectual and operational acceptance and absence of successful military sensitivity attacks can in part be attributed to this National emphasis on data system security. Embedded systems are apt to contain classified information to be protected from disclosure. This information can include targeting information, high resolution terrain data, attack plans, characteristics of the system itself, secret keys, or access control information related to individual users.

Encryption

The window of vulnerability of classified data can be reduced through encryption, leaving the problems of covert channels, key management and key protection. Encryption reduces the need to destroy data in a captured system if critical functionality is left in software which is encrypted when not in use. Cryptanalysis is of concern if the algorithms are not of sufficient strength and usage is not correctly engineered and verified. The key can be a function of time, the event, or the user, where the key is not available except when appropriate. If it is desired to purposely plant false information, release along with an improper cryptographic checksum is a secret way of telling the friendly forces (with the key) that the information is not valid.

Transient Classification

It is often necessary to reclassify or downgrade data unexpectedly. This is sometimes associated with an event such as the DEFCON level. (DEFCON is a national level of alert which includes day-to-day, local crisis, conventional war, regional war, and general strategic nuclear war. The higher the DEFCON level is, the more apt we are to use information that might reveal a secret information source.) The idea of providing classification flexibility in a tactical situation is good, but will almost always provide the perpetrator a way to avoid the security policy. Sanitization rules can be used to qualify data for downgrade by removing or hiding (e.g., statistically) the characteristics that caused the data to be classified. The process must be designed so that a manual reviewer can keep up with speed requirements and so that neither a manual or automatic reviewer can be spoofed. Careful design uses both manual and automatic support, perhaps even using an expert system.

Classified Actions

Often, output of an embedded system is not printed data but rather electronic data that control electromechanical components of the host system. An embedded system that uses classified information for decision making may reveal sensitive information through actions. Presence, identification, and authentication of observers may be required. Communications with human users might be through synthetic voice or graphic images. The information medium or another medium must convey classification, but in either case, it must be conveyed indisputably and in such a way to be taken and remembered in the proper context. Under a Trojan horse attack the control of functions of an electromechanical system may provide a covert channel for leaking classified information. TEMPEST sources may also be covert channels.

Audit Functions

There is a new real time role and functionality for the audit function. The first is that of not only collecting key parameters, but comparing them against history, statistics, expected attack parameters, or conditions that warrant surveillance. Secondly, audit data are fused with one another to increase the amount and quality of information known concerning an anomoly and can be used in real time as a detection mechanism. Detection of illegal entry may still allow time to take action before a sensitivity attack is successful.

Criticality

It is the criticality of a military system to human life or National/military objectives that drives the expenditure of funds to provide integrity and service assurance. Since we are concerned with the insider threat, accidents with high impact are also prevented by mechanisms conceived against malicious threat. We are concerned with trust of individuals involved with the system and recommend much the same clearance approach as for military classified information protection. Integrity concerns such as program correctness, hardware reliability, data precision, and human competence are equally important to critical systems, but are dealt with here only as a byproduct to malicious attack prevention and must be considered as separate system design objectives.

In a critical system criteria [2] we have chosen the Biba model [3] and supplemental proven mechanisms as a basis for criticality because the Biba conditions can be implemented [4] by little more than replacing the Bell-La Padula model in the Orange Book. Thus the designer, implementer, and user community can use their sensitivity background to understand criticality. The Biba approach is cheaper to implement because of the sharing of common mechanisms with sensitivity and the use of less expensive detection/correction mechanisms. Implementation reduces the window of vulnerability by defining distinct security domains, restricting the data flow (via Biba), restricting access to higher critical functions based on trust, need-to-modify, and need-to-execute. In sensitivity, the Bell-La Padula policy must be violated for Top Secret functions to command lower classified elements. In this case we use manual or tightly controlled methods. The same can be done for Biba in an integrity architecture. A Biba model weakness is that it does not prevent Trojan horses from being introduced at the lowest level of criticality, and all software, data and processes at the higher levels must be trusted. In a sensitivity Trojan horse attack, there must be a leakage path. In a criticality attack, the existence of the Trojan horse suffices to do damage. The advantage is that there is usually time for detection/recovery to occur.

A Clark-Wilson model [5] supports a three way policy between user, process and data (or resource). A similar implementation can be achieved with a Biba model and a definition of what programs under different users must do. Expert system auditing functions determine this empirically by building a history and sensing deviations. Such capabilities can augment a design requirements definition of what should happen and what should not happen. This is especially true for availability, which if defined as a requirement under all conditions becomes an integrity problem. Otherwise resource usage can be monitored with a recovery action initiated for abnormal usage.

Relationship with Availability and Survivability

As discussed in [4] and shown in Figure 4, there is a close relationship between criticality (integrity and service assurance), reliability, and survivability. There should be serious consideration toward expanding the risk model to encompass the three areas. The denial of service detection mechanism will detect equipment failure and the probability of data system attack will depend on the potential success in more forceful attacks. An overall mechanism cost savings and increased effectiveness will result if the combined objectives are considered.

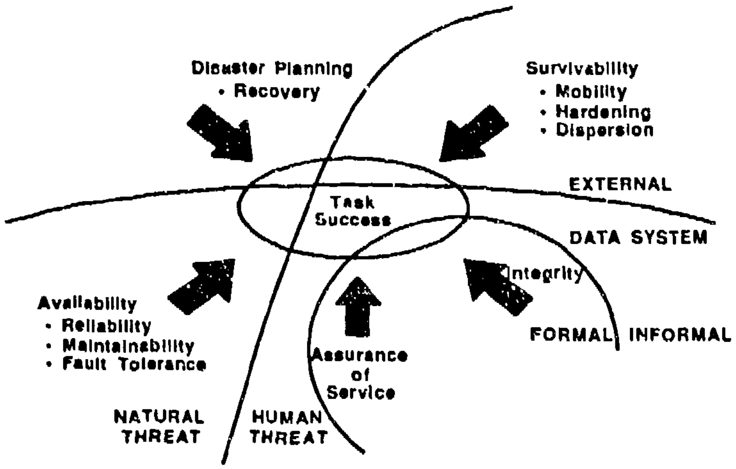


Figure 4. Design Relationships

System Programmability

Programmability is an important factor in the ability to launch a data system attack; however, the lack of ability to program does not preclude such an attack. A Trojan horse can be implanted during the development cycle and maintenance capabilities almost always allow probing and substituting functionality. A maintenance person could easily design and implant a Trojan horse into most systems. Even in autonomous (e.g., robotic) systems, there may be many different applications oriented languages to support the functionality. These could be AI, expert, robot control, or special object oriented languages developed for the user interface.

Initialization, Shutdown, Quiescence

The NSA Blacker program incorporated the concept of a special initialization/shutdown module to be used under security officer control that provides necessary identification, authentication, and initialization data (encryption variables, and access control information) to ensure high integrity transitions between secure and insecure system states. This includes planned, accidental or malicious shutdown or when components are maliciously removed. A similar approach is suggested for embedded systems, especially to place a system in a state of quiescence during storage and transport and bringing it to life at deployment.

Cryptographic Checksums

The window of vulnerability for a criticality attack can be reduced by use of cryptographic checksums and a checking mechanism used sufficiently often to preclude the possibility of a successful integrity or denial of service attack. The difficulties are regenerating checksums with data change, key management and protection. Techniques and concepts that prevent replay and deal with potential noise are helpful, and can be developed using public key cryptographic approaches. Any checksum generated at a high level can be verified at any level using public key methods. This is key to critical system design.

Detection and Recovery

Detection and recovery within a specified critical time is an alternative to prevention mechanisms that may be significantly cheaper and more effective. Recovery can be automated, have a man in the loop or be a combination. It can include checkpoints to restore a previous system state. These are an extension to the concept of audit. The trend, especially useful in denial of service attacks is to seek abnormal behavior or departure from historically established norms. General or specific user activity, functional activities, or resource usage can lead to a threshhold of normal and abnormal usage. Any normalcy rules can be included in the design, with adjustments to preclude false alarms or tighten the alarm criteria. Detection can often be a parallel function, not affecting performance.

Criticality Covert Channels

Covert channels allowing criticality attacks are nonnormal paths through which malicious code may be entered and executed. Input from sensors, maintenance functions, any external communications potentially allow code to be inserted and activated.

Complexity

There are three aspects to complexity; interface complexity due to multiplicity of components and non standard interface definition; policy complexity where different components have different security policies creating the need for a policy between any communicating components; and temporal complexity where components were built at different times for different reasons and where requirements, threat, and mechanisms have evolved. N component elements can have an N**2 connectivity problem and an N**3 communication path problem. A state transition model does not work in a parallel and distributed environment without overconstraining the system. Where different policies exist, the cascading problem must be considered. Multiple security models are difficult to interface and may be invalid. Reference [6] addresses these issues.

Produced in Multiple Units with Wide Distribution

The Orange Book assumes the existence of a facility to house the system. There is often no facility for embedded systems except the hostile world. To reduce the window of vulnerability, varying levels of protection can exist for different components, but not with a great deal of thought as to the resultant risk of the combined system. Field assembly must be accomplished under special conditions with specific procedures.

A piece of tactical equipment may exist in storage for a long period of time before being called into operational use. It may be occasionally deployed as a part of an exercise, but in either case has no continual monitoring as in the case of other secure computer elements. Changes are often required on a mass basis and must be accomplished under the intended secure level of the use of the equipment to avoid implant of a Trojan horse.

There is a need for configuration management and secure delivery and placement. Classified or highly critical electronics and programs may be controlled as separate items for last minute insertion into the system before its use. It must be assured that an attack cannot be launched from interfacing components.

Partial or Total Autonomy

A semiautonomous tactical system may be mobile and concealable, operate on the move, be survivable, endurable and robust, be essentially quiet except for needed communication, and be dispensible in an operations (since battle planning anticipates attrition). Operating components are standardized with high reliability, assume little or no field maintenance and degrade gracefully. In a gracefully degrading capability the system must quiesce when it becomes insecure.

Human Security Functions

Standard computer systems have a cooperative human to assist in security, unlike an autonomous system or one that merely supports a human function (e.g., pilot). A human understands what is classified and its manifestations. He/she considers the relationship of this information to his/her actions and assesses when it is communicated and when it is not. Actions will then be based on audience and environment. Whether communicating with words or actions there is an identification, authentication, and labeling process required, that is, passing on to others the fact of the classification.

Humans mask their actions if they think they are revealing classified information. They employ deception strategies. An embedded system that operates autonomously must act under these same considerations, at least to the extent that techr ›logy allows, or otherwise be in a secure quiescent state. A pilot once he receives his target location may approach it via a circuitous rout. A remotely piloted vehicle must be programmed to do this same. An autonomous system has an automatic audit function with self initiated action (e.g., alarm or shutdown with data erased or encrypted).

Trusted Path

Just as we presently define a trusted path between users and the trusted system base, there must be a trusted path between the system and its sensors and effectors to help fight against jamming, spoofing and replay. There must be continuous knowledge by the components of who they are connected to and that a proper and secure interface is maintained.

Unfriendly/Hostile Environment

A hostile environment is one outside a computer facility. The worst case is when the enemy has unconstrained access and is assumed to have technological skills and extensive physical might. Security may include strong packaging, and ruggedization for extreme conditions.

Relationship between Data Systems and Physical Security

There is a growing relationship between data system security and physical security. Computers interpret sensor output, identification devices, and area surveillance. Encryption can be employed in data storage instead of a guarded room. The concept of fusion can mean the joint interaction between a guard and an identification device. Cryptographic checksums can help assess the activities of a suspected intruder. Insider and outsider threats must be considered by the designer when planning system protection mechanisms.

Security Authority

Traditionally security is the reponsibility of the military police or intelligence function. These roles are usually not trained in the complexities of data system security and often necessary clearances are not given. If security is going to be part of the tactical environment then an authority must exist that reviews data and takes effective action. The activity required by a security authority should be minimal during a battle or crisis situation since attention is turned elswhere. The authority must report high within the command so there is an understanding for the relative importance, the actions and tradeoffs. The design of the system must consider who the authority will be and what can be expected in terms of attentiveness and response. How will a mobile or autonomous unit communicate data to the security authority? A patch-in capability, a removable cartridge or even an RF connection are all possibilities, coupled with security authentication and spoof detection.

Identification and Authentication

The future battlefield will be characterized by high attrition in personnel and equipment. New replacements and allies might be placed in a position of assistance. We are perhaps on the threshold of unique electronic dosiers for each individual based upon characteristics and features. The database of identification information for all possibilities would be very large and unwieldy unless massive compression, or massively large storage and search techniques can be employed. Less satisfactory techniques, such as smart cards, passwords and keys, must be used in combination to gain any degree of assurance.

Continuous or almost continuous identification/authentication of either person or component is critical, such as reauthentication through a continuous mutual dynamic key. In the case of humans the monitoring of key stroke (including pressure and timing) or other predictable human interaction (e.g., voice) can be used as an authenticator.

A battlefield situation cannot withstand delay resulting from a security problem. The capability must exist for two person sponsorship or override by an unforgeable identifiable higher command authority, especially one backed up by a coauthority. Ultimately the commander must be presented with the evidence and be able to decide what is to happen in a battlefield situation and not be controlled by security mechanisms.

Capture and Duress

There must be a way in which a user is able to communicate a duress signal without revealing in any way the fact that he has done so. The fact of withholding full capability must be reviewed by a higher authority. A deception plan for duress must be available and put into place. Likewise a captured system must be able to notify other systems of its captured state and its capability must be altered and, if desirable, an alternate mode of operation employed. Sensitive unencrypted data need to be destroyed and functionality needs to be disabled. The mechanisms must not present a denial of service risk.

Response-Driven Operations

Many embedded systems are response-driven. Realtime systems must operate under the constraints of a clock. Functionality is carefully engineered based on statistical or known input to ensure that functions will be completed in time for the usage of output data. Other response driven systems may not be as periodic or predictable, but on-time responses are vital. The approach is to identify functionality, including security before doing a functional allocation, and identifying response requirements. Fortunately, many security functions are not real time and can be achieved in parallel, as long as the long term computational requirement is less. This can be achieved through parallel and pipelined design.

Summary

Here we have addressed topics on the security of an embedded systems. We have encouraged the expansion of historical treatment of security because of the suitability and applicability of the formal approach to development that has been a part of secure systems technology. This expansion takes security beyond the current responsibility of some agencies (e.g., NSA) and into the realm of (e.g., military) operations.

This paper suggests a hierarchy of requirement/criteria and guidance documents shown in Figure 5. The Orange Book [1] stands as the standard for formal security policy. It must be augmented by a Trusted Critical Computer System Evaluation Criteria (e.g. [2]) where either or both may be applied in a development. In the future these documents can be combined into one applicable to many different policies (Biba, Bell-LaPadula, Clark-Wilson).

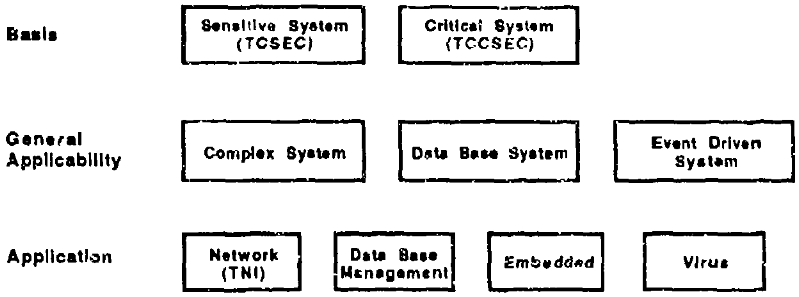


Figure 5. Security Guidance

At a second level should be interpretations with wide application. Part of the Draft Trusted Database Management System Interpretation applies to database management while part is applicable to database management systems. The former should be isolated to the second level interpretation with the latter adopted as an application interpretation. Other second level documents should be written on Trusted Complex Systems (e.g., [6]) and on Trusted External Event Driven Systems.

The third level documents can interpret the higher level documents for specific applications such as networks (e.g., TNI [7]), database management systems (e.g., TDI [8]), embedded systems (e.g., draft TESI [9]), and microprocessor systems. It also appears that threats could be addressed at this level such as the virus threat, telecommunications threats, and the Trojan horse threat.

Graphics for this paper were by Donna Henry of CTA. The authors appreciate review comments and discussion, especially from Trusted Information Systems, Inc.

References

[1] DoD 5200.28-STD, "Trusted Computer System Evaluation Criteria," December, 1985

[2] Johnson, H.J., "Trusted Critical System Supplement to the Trusted Computer System Evaluation Criteria," DRAFT Information Intelligence Sciences, Inc., Delivered to Computer Technology Associates, December, 1988

[3] Biba, K.J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April, 1977

[4] Johnson, H.L, "Security Protection Based on Mission Criticality, Proceedings Fourth Aerospace Computer Security Applications Conference, IEEE, December 12-16, 1988, pp.228-232

[5] Clark, D.D, and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the 1987 Symposium on Security and Privacy, Oakland, CA 1986, April 1987, pp. 184-194

[6] Johnson, H.L., "Trusted Complex System Interpretation to the Trusted Computer System Evaluation Criteria," DRAFT Information Intelligence Sciences, Inc., Delivered to Computer Technology Associates, December, 1988

[7] NCSC-TG-005, "Trusted Network Interpretation," 31 July 1987

[8] National Computer Security Center, "Trusted DataBase Management System Interpretation," DRAFT, May, 1989

[9] Johnson, H.L., "Trusted Embedded System Interpretation to the Trusted Computer System Evaluation Criteria," DRAFT Information Intelligence Sciences, Inc., Delivered to Computer Technology Associates, December, 1988

This work has been partially sponsored under CTA Contract Number F41621-88-05001 for the U.S. Air Force, Hq. Electronic Security Command/LGCCV, Kelly AFB, TX 78243. The opinions are only those of the authors.

A "HOW TO" GUIDE FOR COMPUTER VIRUS PROTECTION IN MS-DOS

M. H. Brothers

AT&T Bell Laboratories
Whippany, New Jersey 07981

The proliferation and casual administration of personal computers has created a potential Achilles' heel in today's computer operations. The following procedures cannot prevent or protect a computer from all attacks in the future, but can serve as a guideline for safe computing in the current environment.

Review of Terms

We first need a shared understanding of common terms required to discuss virus detection, recovery, and prevention.

The Disk

The formatted disk has a number of physical tracks created for the orderly storage of data. Each track is subdivided into sectors, with logical numbering of both tracks and sectors. The boot track is typically the first track on the disk, containing the start-up program that is executed when the PC is first powered or restarted.

The next important section of the disk is the File Allocation Table, or FAT, a secondary index that points to subsequent clusters in an accessed file.[4] The first cluster, or beginning of the file, is listed in the directory structure within the operating system. If the FAT were disabled, all stored data that spans more than the first cluster would be unreachable. The FAT consists of pointers, or entries, for each cluster on the disk. The pointer could indicate:

1) The cluster is unused.
2) The cluster is damaged, marked as a "bad cluster."
3) The next cluster in a given file, creating a linked list.
4) No more clusters associated with a specific file.

Both the boot track and the FAT are common attack points for destructive software. The procedures outlined in this document aid in protecting these crucial components from corruption.

Booting

Starting up a PC, or "booting," can be performed in two different modes. In a "cold boot," the PC must be physically turned on. If the operating system is resident on a hard disk, then just providing power starts the boot sequence. If the operating system is resident on removable media, then the media, in this example a floppy disk, must be placed in the floppy drive before the machine is powered for the boot process to take place.

The second way to start a PC is when the machine is already running. The term used is a "warm boot," and can be performed in one of two ways. For some PCs, simultaneously pressing the ALT-CONTROL-DELETE keys causes the operating system to re-initialize. RESET will also re-initialize the system, in addition to running the self-diagnostics and clearing the volatile memory.

In booting, the DOS operating system uses two hidden files and three visible files. Prior to any file, the boot record is activated. The boot record, usually resident on side 0, track 0, sector 1 of the disk, contains the basic information about the disk needed by the operating system. From the boot record, the PC then seeks the first hidden file. BIO.SYS (file names will vary with the operating system), a file that assumes control of the PC from the operating system and continues the loading sequence. The BIO.SYS loads MSDOS.SYS to introduce enough intelligence to the PC to load COMMAND.COM, the first overt system file. COMMAND.COM contains the command interpreter program that serves as the interface between the person at the PC, the rest of the DOS operating system, and the PC hardware. Through COMMAND.COM, the PC user can access the internal DOS commands from any directory. These three files must be present, in specific positions on the disk, to successfully boot the PC.

The other two visible files, CONFIG.SYS and AUTOEXEC.BAT, perform other duties for the operating system and the PC. CONFIG.SYS contains instructions that configure the PC. The CONFIG.SYS can include setup instructions for a RAM (Random Access Memory) disk, or instructions for accessing remote disk drives on a LAN (Local Area Network), for examples. In AUTOEXEC.BAT, the operating system has a special batch file that instructs DOS to execute a series of commands once the PC has finished booting. The AUTOEXEC.BAT file is usually created by the user or the administrator who sets up the PC to the user's specifications.

All of the files just mentioned, because they are automatically accessed by the operating system when booting, are primary targets of destructive software.

Protection and Prevention Procedures

A computer under attack by a virus may manifest symptoms identical to a hardware failure. Following these procedures will minimize the end user's vulnerability to a computer virus and will also serve to minimize the negative effects of a hardware failure.

Write-Protection

Always use write-protection on removable magnetic media, such as floppy disks. Only remove the protection when a specific write to the medium is required. This practice will only protect the floppy disk when the write-protect tab is in place. Removing or disengaging the write-protect tab leaves the floppy disk vulnerable to an unauthorized write. By using write-protection on your removable media, however, you have introduced an early indicator of potentially unauthorized write attempts originating from your environment, possibly from the software already installed on the hard disk. Extra write-protect tabs are supplied with new floppy disks by the manufacturer, or you can use any opaque tape.

Write-protection for the hard disk is not a trivial matter at this time. Access control packages for the DOS environment currently exist that can partition the hard disk into open and write-protected sectors, or will assign an access level to each resident file, whether data or executable. Recognize, however, that the sector write-protection cai be subverted by direct virus attack. Encryption of the hard disk can make unauthorized file modification difficult. In theory, the entire operating system of executables should be protected from unauthorized writes, yet be capable of handling legitimate operating system updates.

Introducing New Software

Introducing new software is always a trying time. Having to worry about hidden viruses in the software can prove to be too much of a burden for the typical user. Here, then, are some guidelines to determine when to be worried about newly acquired software.

Newly Acquired Software	
Low Risk (Less Worry)	**High Risk** (More Worry)
You've paid for it (legal liability).	It's free.
Your software source is an old, trusted supplier of your software needs. Your supplier is well-known, widely used, and has a good track record of supplying quality products with good "after-the-sale" support.	Your software was downloaded from a public bulletin board (public domain software, even vaccines!). A new software package arrives by surprise in the mail from an unrequested source. A copy of a program is acquired from your friend, your neighbor, your relative, or your co-worker.
Your newly acquired software is a well-known package, commonly used by many of your peers.	Your newly acquired software is unknown to you and your peers. No performance track record.

When downloading data from another computer, always download to a floppy disk instead of to a hard disk. Use the DOS **CHKDSK** command to check for hidden files.[7] If the disk has a label, you can expect one hidden file. Verify that the label is present by using the **DIR** command if a hidden file is indicated. Whenever possible, share only source code, not object code.

The following general guidelines can be used for introducing any new software that doesn't fall under the "high risk" classification.[5][4]

Introducing New "Low Risk" Software	
Floppy disk system	**Hard disk system**
1. Write-protect system diskettes. 2. Use expendable diskettes. 3. Scan for tell-tale messages in text strings.	1. Perform a full back-up. 2. Perform checksums on all resident files. Run the new program. 4. Run the checksum program again, compare.

Use the following procedures, in the order listed, to minimize unwanted surprises from "high risk" software:

1. Verify the authenticity of the software with the supposed source. When ordering the software, discuss with the supplier some means of incorporating a unique identification code in the documentation or software program that can be checked for accuracy upon receipt. Ask your software supplier for certification of virus-free code.[1]

   In general, always use software from reliable sources. If you must use public domain, shareware, or freeware programs, contact the writer or distributor and compare the file date and file size before using the program.[7]

2. Before running the software, do a complete system back-up and verify that you can recover from the back-up before you load the new software. The first line of defense against a software virus will always be a full and adequate back-up.[4]

3. If the software is being generally distributed, wait one month before loading it (a "soak" period) and watch the news networks for chatter about bugs in the software.

4. Run the software on an isolated machine; drop all network lines, either physically (preferable) or logically.

   Always quarantine your test machine. Quarantined machines only use quarantined disks, disks that are not shared with any other machine.[4]

5. Make sure the program is running properly with no hidden activity. One way to check for hidden activity is to load your software on an expendable hard disk and then reboot the system from a write-protected floppy system disk. Never put shareware or suspicious programs in a hard disk's root directory; most viruses can affect only the directory from which they are executed.[5] If running the software causes a write error to the floppy system disk when no write to the system disk was expected, further investigation is needed. Not all virus programs cause write errors, but this is still one of the most common ways for a virus to fail in the attempt to replicate itself.

   Run the new program under a variety of system dates to check for a date-triggered logic bomb.[1] Try the following dates[5]: one month ahead of the current date, one year ahead of the current date, the next Friday the thirteenth, April first, October 31st

6. Again, allow a "soak" period of isolated activity with the new software resident before reconnecting the networks. Watch for unexpected write-errors, changes in the operating system, file size changes, and generally anything unexpected or different from your normal operation. Keep a manual log of file sizes and check against current file sizes for unexplained "growth."

Limiting Machine and Media Access

1. Introduce password protection as access control to your computer. For DOS-based desk-top computers, firmware-based access control is currently the most difficult type of protection to compromise, but a physical lock must also be introduced to protect the new circuit board from modification or theft.
2. Lock up all removable media when the media is not in use. A small, locked file box on top of your desk is not sufficient, due to the box's portability; put the file box in a desk or cabinet drawer that is also lockable.

Back-Ups

Back up your system on a regular basis, and make sure that you can recover from the back-up media!

1. Three generations of complete system back-ups are strongly recommended for the individual user's computer. Each "generation" is a complete back-up of all data files, both on hard disk and removable media. Application and system files need only the master copy, a working copy (on either floppy or hard disk), and one back-up copy to be considered secure. An additional back-up copy of application or system files is needed when modifications or updates are made to one of them. Carefully date all back-ups and retain your back-up records for at least one year.[5]
2. All back-ups should be secured. When determining the risk for your machine, consider the possibility of a general emergency barring any employee from entering the entire building. Do you need to be able to grab your back-ups from another site and rebuild your system, or can you wait for your own building to be reopened?
3. In between full system back-ups, updates of critical files can be stored on flexible media, properly marked to indicate the sequence of retention. Full back-ups have to be planned by the user and cannot be dictated as a standard time interval. Critical files should be backed up whenever updated. Critical systems should be backed up whenever accessed, with full back-ups done daily if accessed several times a day. At a minimum, your PC, if it contains data files, should have a complete back-up done once a month.

   Many software packages exist to make the job of backing up easier. You can consult *PC Magazine*, volume 6, number 8, dated April 28, 1987, for an evaluation of commercial packages.

Specialized Software

Many commercial vendors now offer "vaccine" programs, software designed to limit your computer's exposure to virus programs. Most of these vaccines work by thwarting known modes of penetration of your computer's files by today's viruses. A few vaccines claim to use artificial intelligence, enabling the vaccine to learn from new viruses that attempt to invade the computer. Consider, however, that researchers cannot isolate all of today's viruses for detection software.[1]

Consider the installation of a vaccine with great caution. Some hackers have been known to offer a vaccine, especially on the public bulletin boards, that turns out to be a virus itself, designed for harm. The best defense against this contemporary threat is user awareness and safe computing habits.

Software diagnostics will aid in the detection and prevention of infection.[2] A simple checksum program could aid in the early detection of changes to supposedly stable files. Checksum programs are usually written as subroutines within diagnostic programs. In principle, the algorithm will add the number value of each file byte along with a weighting factor to create a single value representative of the entire file. Comparisons of the file's checksum to past values will aid in the detection of changes to the file. Cryptographic checksums should be used, when possible.

Recycling Media

When recycling a floppy disk, always use the DOS **FORMAT** command to reformat the disk; DO NOT simply erase all the files from the disk.[7] Remember not to share disks for quarantined machines and don't accept disks from unknown or untrusted sources. Reformat all empty disks given to you, just as a precaution. Special software exists that can do a thorough erase of computer storage media.

Sharing Files Safely

When transferring files on a floppy disk, place the output data on a floppy that has no executable files, including system files. Arrange in advance with the intended receiver of the data a handshake system to verify the authenticity of the data received.

Detection Procedures

Detection procedures for software virus activity may unintentionally identify TSR programs as suspicious. TSR, or Terminate and Stay Resident, programs are pop-up programs that jump in and out of the front process as needed. Most TSRs "hook into" an interrupt vector before they go TSR. These hooks might interc pt and process key strokes, or "hot keys," or they might hook and intercept direct disk writes themselves.[4]

Unusual Activity

Viruses may affect the complexity characteristics and size of infected programs and, as a result, become detectable or harmful.[1] Look for unusual activities to detect a virus, such as:[7][5]

- An unexpected attempt to write to a write-protected file.
- An unexpected change in the size of one of your programs or a sudden decrease in overall system free space.
- A change in the last date of an executable file's access. The change could be due to either a modification or an update to the file. Also notice if several executable programs have all suddenly changed the date of last update to the same day.
- Diagnostic errors, from your special utilities, like an unexpected checksum discrepancy or a change in the image of the system interrupt vectors.
- A change in the normal system behavior. Examples would be an increase in the number of "lost" files, a change in the rate of media errors, and overt symptoms of a virus attack. Overt symptoms can include strange messages on the monitor ("Ha, Ha, Gotcha!!"), a change in your PC's mechanical operation (the cursor takes on a life of its own), or numerous unexpected disk accesses.
- The following is a good check-up routine for verifying the health of your system. Please note that command syntax may vary with different operating systems; consult your operating system user's guide. Additionally, any of the commands can be piped to a print file or piped through a **MORE** command to display the output, one screen at a time.
  - Periodically use **CHKDSK** to check your DOS directory, watching for changes in the number of hidden files.

    CHKDSK [d:] [filename] [/F] [/V]
  - Maintain an up-to-date hard copy of your directories and their contents. Use the DOS **TREE** command to print the directory structure. (A similar command from a special utility will also work.)

    TREE | MORE
    Displays current directory files and subdirectories.

    Use DOS **DIR** to print complete information about each subdirectory's contents. Watch for unexpected changes in file size and for the appearance of new files.

    DIR | MORE
    Displays directory in original form.
  - Use the DOS **SORT** command or its equivalent to sort each subdirectory by date and time. Any date before 01/01/80 should be suspected. Future dates, like 01/01/01, should be carefully checked. Check any date that contains 00 or any time later than 23:59:59.

```
DIR | SORT /+24 | MORE
```
Sorts directory by month.

```
DIR | SORT /+31 | MORE
```
Sorts directory by year.

```
DIR | SORT /+33 | MORE
```
Sorts directory by time.

- Sort each subdirectory again, this time by file size. Watch for unusually large files or files with a size of 0 bytes. Any inexplicable size change in COM, EXE, BAT, or SYS files should raise a warning flag.

```
DIR | SORT /+14 | MORE
```
Sorts directory by file size.

- Now do a subdirectory sort by file name. If one of your files is called 123.EXE, for example, and you find a file with the same name but the file extension of .COM, this could spell serious trouble. A COM file executes first in the DOS hierarchy. Also check for odd or unfamiliar file names.

```
DIR | SORT /+1 | MORE
```
Sorts directory by file name.

- Finally, sort each subdirectory by file extension. You may not have picked up suspicious-looking files or extensions, such as DBASE.EVL or 123.WK8, the first time through.

```
DIR | SORT /+10 | MORE
```
Sorts directory by file extension.

You Suspect a Virus

If a virus is suspected, take the following actions:

1. Leave the machine running! Any evidence of intrusion or infection may be lost if the machine is powered down. In the haste to restore the system as quickly as possible, many clues are often overlooked and even destroyed.[6] Turn off the machine only at the instruction of your management, your security group, or your technical support.
2. If your desk-top computer is connected to any kind of network, break the network connection. Break the network connection either physically or logically. A physical break would mean pulling the plug on all network connections and is the preferable procedure, if your action would not bring down the entire network. In no case should you continue to use your network facilities with a potentially compromised machine.
3. Let people know about your suspicions. Alert your own management.
4. Use your regular trouble reporting procedure to notify technical support of your problem. Your technical support may be official or unofficial. On your support person's advice, do a complete back-up of your fixed media to clean, formatted, removable media for later

analysis. Do NOT use your suspected computer to format the needed flexible media. The FORMAT.COM is an executable routine and could have been compromised by the virus on your machine. If your disks need to be formatted, use an uncompromised machine. Then, once the back-up of the contaminated system has been completed, proceed with the recovery procedures outlined below.

Although this set of procedures addresses the very real concern of computer virus activity, do not assume that every computer failure indicates the presence of a virus. In the event of a computer malfunction, take reasonable steps to ensure the safety of other machines, and proceed with an orderly analysis of the situation.

Recovery Procedures

Reboot

Reboot your machine from a write-protected, uncontaminated copy of your system software (DOS). Referencing the drive containing the clean copy of DOS, reformat the contaminated hard disk. In a multi-partitioned hard disk with non-DOS partitions, a low-level format is recommended to ensure the removal of any contamination. The FORMAT.COM routine must reside on your trusted DOS source. The reformat followed by a complete power down should wipe out any contaminant. The power down cleans the volatile memory of any programming remnants.

Rebuild

Rebuild your hard disk from a trusted back-up. If you have the time and inclination, you can work your way back through the most current back-ups, loading each one in turn and checking for the identified contamination. If the back-up appears to be contaminated, then you will have to do a complete reformat again, from the tiusted DOS source, and start building your system all over again. If you want to minimize your time and effort, then go back to your original application software back-ups, on write-protected media, and rebuild your system without any data files, just like the day it was installed. In either approach, do not reconnect to any network that you might have available until you are sure that you have a clean machine.

Network Considerations

Shareware

In local area networks, LANs, avoid placing shareware in a common file server directory. Such placement would make the shareware accessible to any PC on the network. Only the network administrator should have the ability to sign onto the file server node.[5]

Virus Manifestation

If a virus were to manifest itself on a computer network, the administrator may be able to identify its presence through a change in the type or frequency of trouble reports.[3]

Network Guidelines

As with individual computer systems, the ultimate defense position for the computer networks is to perform back-ups. The following are guidelines for keeping your network healthy, for any PC on a network.[3]

1. Write-protect the boot medium.
2. Limit network users' network access to an "as-needed" basis.
3. Maintain several generations of back-up tapes for the central file server, if applicable. See Section II.D.
4. Do not use new programs, or updated versions of existing programs, unless they have been in public domain for at least four weeks.
5. Use diagnostic software to check programs for viruses.

References

[1] L. M. Adleman, "An Abstract Theory of Computer Viruses," University of Southern California, 1988.

[2] "Antidotes and Hype," Information Center, p. 41, 1988.

[3] R. Bunzel, "Flu Season," Connect, pp. 40-42, Summer, 1988.

[4] R. M. Greenberg, "A Form of Protection for You and Your Computer," 2600 Magazine, Issue 4-7, pp. 28-38, Summer, 1988.

[5] M. Hahn, "Protecting Your PC Systems from Computer Viruses," Computer Security, Number 87, pp. 1-2, May/June, 1988.

[6] H. J. Highland, "Random Bits & Bytes," Computer & Security, No. 7, pp. 3-11, 1988.

[7] "Protecting Against Computer Viruses: Know Your Enemy," Lotus, pp. 17-18, July, 1988.

[8] L. Roshfeld, "Journey Through DOS, Part 1," Lotus, pp. 88-93, October, 1987.

THE "FATHER CHRISTMAS WORM"

by

James L. Green
National Space Science Data Center
Goddard Space Flight Center
Greenbelt, MD 20771

and

Patricia L. Sisson
SPAN Security Manager
Science Applications Research
Lanham, Maryland 20706

June 1989
Submitted to the 12th National Computer Security Conference

ABSTRACT

Three days before Christmas 1988, a computer worm was released on a very large international DECnet network. The worm reproduced itself and was received on an estimated 6,000 computer nodes worldwide. However, only a small percentage of these nodes actually executed the program. The computers that successfully ran the program would try to propagate the worm to other computer nodes.

The worm was released onto the DECnet Internet from a computer at a university in Switzerland. Within 10 minutes after it was released, the worm was detected on the Space Physics Analysis Network, or SPAN, which is NASA's largest space and Earth science network. Once the source program for the worm was captured, a procedural cure, using existing functionality of the computer operating systems, was quickly devised and distributed. A combination of existing computer security measures, the quick and accurate procedures devised to stop copies of the worm from executing, and the network itself, were used to rapidly provide the cure. These were the main reasons why the worm executed on such a small percentage of nodes.

The purpose behind the worm was to send an electronic mail message to all users on the computer system running the worm. The message was a Christmas greeting and was signed "Father Christmas." This paper presents an overview of the analysis of the events concerning the worm based on an investigation that was made by the SPAN Security Team and provides some insight into future security measures that will be taken to handle computer worms and viruses that may hit similar networks in the future.

INTRODUCTION

The Space Physics Analysis Network, or SPAN [1], has been an extremely reliable international scientific computer network that has become a major element in NASA's quick reaction capability for supporting many major NASA missions over an eight-year period [2]. Major features of SPAN are its ease of use, efficiency, and availability to scientists conducting research in scientific disciplines such as astronomy, astrophysics, climate, Earth, ocean, planetary, life, and solar terrestrial science.

Currently, SPAN ties together well over 2,800 computers at NASA centers, other government agencies, private companies, and universities in the United States, with extensions to the European Space Agency's E-SPAN network. SPAN utilizes computer-to-computer communications (DECnet protocol) allowing mail, binary file transfer, and remote log-on capability. The majority of the computers connected to the network are VAX machines running the VMS operating system. SPAN is managed by the National Space Science Data Center (NSSDC), located at NASA's Goddard Space Fight Center (GSFC).

SPAN has interconnections with several national and international wide area networks such as HEPNET, INFN, THEnet, DAN, GEONET, UARSnet, and ASTRONET. All these networks cooperatively manage unique computer DECnet addresses. The nodes from all these networks then form one transparent worldwide network called the DECnet Internet. The combined total number of computers reachable over the DECnet Internet is about 12,000. To the user, the DECnet Internet operates like one "easy-to-use" network. The DECnet Internet, on one hand, has solved the problem of transparency between computers regardless of what DECnet network they are connected to. On the other hand, the DECnet Internet provides the connectivity to make one network's security problem everyone's concern.

On December 22, 1988, at approximately 17:00 EST (eastern standard time), a computer worm was discovered on SPAN. This worm has been affectionately called the "Father Christmas Worm." A computer worm is a program that is self-contained and has the ability to propagate itself across a computer network to any idle machine. Unlike a virus, a worm does not modify another program. In the case of the Father Christmas Worm, virtually any computer on the DECnet Internet could have received a copy of the program. However, an individual computer may not have the system software config[illegible]on that would enable it to execute the program (because of the implement[illegible] certain security precautions).

The purpose of this paper is to provide an overview analysis of the events concerning the Father Christmas Worm, based on an investigation made by the SPAN Security Team. From this investigation it has been determined that the worm was released from a computer (node number 20597::) at a university in Switzerland. Much of this analysis would not have been possible without the extensive help and assistance of the system manager of node 20597::.

The Father Christmas Worm was designed to travel quickly. Estimates are that it was copied to over 6,000 computer nodes. However, it is believed to have executed on only a fraction of those computers.

HOW THE WORM WORKED

The worm program was named HI.COM. The COM file type in VAX/VMS signifies a command file and is usually written in the DEC command language (DCL). DCL provides a user with access to operating- and network-level system functions on a local or a remote host.

Figure 1 provides a graphic overview of how the worm propagated and executed on other nodes. During execution of the worm, Node A transferred the worm file (HI.COM) to Node B. Node B was determined by a section of code in the worm program that randomly generated node numbers and then checked to see if the node was reachable. Once the transfer of the program was complete and Node B had the worm, Node A then would try to direct Node B to execute the HI.COM program.

So long as the worm was executing, it would continue to search out randomly reachable computers and try to propagate itself. On the DECnet Internet, separate blocks of DECnet addresses are allocated to individual wide area networks that are not confined to geographic regions. The use of randomly generated node numbers by the worm program would ensure a worldwide distribution across many networks, which would increase its survivability.

Node A would try to execute the worm on Node B by one of the following two methods:

- TASK Object 0 - If a system level program, called TASK Object 0, is installed in a VAX/VMS computer, it will accept and execute commands from another computer. In other words, TASK Object 0 allows task-to-task jobs to be run between two computer systems. In the case of the Father Christmas Worm, nodes that were following the SPAN security guidelines had TASK Object 0 disabled and were not able to execute the HI.COM file. In addition, nodes that had disabled TASK Object 0 would also not propagate the worm.

- Username/Password Combination - Another way to direct a remote node to execute a program (in this case HI.COM) is by providing a legitimate username/password combination for verification by the remote node. The Father Christmas Worm also tried a username/password combination of DECNET/DECNET. This combination of username/password has strongly been discouraged from use in documentation by the Digital Equipment Corporation (DEC) and in the SPAN Security Policy and Guidelines document [3].

In the example shown in Figure 1, Node B had TASK Object 0 installed. Node A then directed Node B to load HI.COM in memory and, disguising it under the process name MAIL_178DC, begin execution. The renaming of the process from HI to MAIL_178DC was done to hide the fact that a foreign program was executing. Mail processes execute quite frequently on these computer nodes and are easily missed by a system manager monitoring the system. Once executing on Node B, the worm deleted the file HI.COM that was stored on the disk, once again covering its tracks. Next, the worm mailed Node B's welcome banner to the remote node/account 20597::PHSOLIDE in Switzerland. This action provided the initiator of the worm a record of the nodes that were able to execute the worm program. However, there is no accurate record of the nodes that received a copy of the worm but did not execute it.

The MAIL_178DC program also went through a series of time checks looking for 1988-12-24-00:00 on the computer clock. If the actual time did not match the Christmas Eve time, the worm randomly generated a new computer node number (Node C in Figure 1). If Node C was operationally available over the network, then the Node B worm networked the HI.COM file from its memory to the new Node C and asked Node C to execute the program. The cycle then started all over again.

If the system clock time on Node B (or any node executing the worm) was greater than 1988-12-24-00:00, the worm created a listing of all the authorized users on that system and sent a Christmas greeting to all those users. The Christmas greeting message is shown Figure 2. It is signed by "Father Christmas." After sending out the Christmas mail message, the worm then deleted the user list it created and stopped execution.

WORM EVENT TIMELINE

On December 22, 1988, at 16:52 EST, the Father Christmas Worm was released from node 20597:: onto the worldwide DECnet Internet. The worm was first noticed at GSFC by John McMahon, systems manager of SPAN node CSDR, at approximately 17:00 EST, some 10 minutes after it had been released. After notifying SPAN management and the NASA Science Internet Project Office (NSIPO), John also contacted GSFC security to register the unauthorized access to U.S. Government computers. The worm command procedure HI.COM was captured at GSFC, as it had been at several other locations throughout the network, and the task of analyzing it began.

The SPAN Security Team sent messages to all SPAN NASA center managers warning them about the worm and what action to take to stop it. The SPAN NASA center managers are responsible for distributing warnings to the remote SPAN sites that are directly connected to them. Notice was also sent to HEPNET and THEnet representatives.

NASA personnel at the Jet Propulsion Laboratory sent out a warning mail message to 20597::SYSTEM on December 22, 1988, at 23:30 EST. The warning stated that the running of an automated command procedure, like HI.COM, was not permitted on SPAN. This message was received but not read, since it was very early in the morning in Switzerland and 20597:: was running unattended, which is quite common.

The PHSOLIDE account (where the worm started) was again logged into on December 23, 1988, from 1:58-2:23 EST. During this time, all the mail messages containing the system banners from the systems which successfully executed the worm were read and deleted.

The DECnet Internet line linking 20597:: to the rest of the world was disconnected on December 23, 1988, at approximately 03:41 EST. This link was scheduled to go down for an upgrade to the circuit. The action had nothing to do with the worm, but it did isolate an active worm on the large local area network at the university in Switzerland, where it continued to propagate to the local university nodes (see next section).

During the early course of trying to stop the worm, several network systems personnel, on their own initiative, issued procedural patches or cures for the worm. It is important to note that, unlike some virus situations, no vaccine software was necessary; a tightening up of existing computer systems security features is all that was needed to prevent a node from executing the Father Christmas Worm. The patches distributed were easy to describe and were issued by, for example, SPAN, HEPNET, DCA, and the San Diego Supercomputer Center personnel. The basic elements of all the procedural patches were:

a) Delete/Disable TASK Object 0
b) Stop Process MAIL_178DC
c) Delete all copies of HI.COM

Many of these patches went out on mailing distribution lists, such as VIRUS-L over ARPANET (a TCP/IP network). The Father Christmas Worm itself was also distributed to everyone on the VIRUS-L mailing list (by person or persons unknown to us). By the end of December 23, the Father Christmas Worm was virtually stopped on the DECnet Internet. In general, procedural patches were reasonably good and provided necessary protection against the Father Christmas Worm.

Within several days after the worm incident, the SPAN Security Team received full cooperation from the systems manager of node 20597::. The systems manager supplied the team with detailed logs and accounting records from his system. In February, a detailed report about the Father Christmas Worm was completed by the SPAN Security Team and was turned over to the appropriate authorities.

RESULTS OF THE INVESTIGATION

After carefully reviewing all of the log-in records to the PHSOLIDE account in conjunction with the system manager of 20597::, it was concluded that a user coming through a particular terminal server released the worm program. The terminal server accesses could have come from one building on the campus or from existing dial-in modems. The director of the university where node 20597:: is located has had every authorized user of the PHSOLIDE account (15 such users) sign a non-involvement statement. This affidavit stated that these users were not responsible for the creation of HI.COM nor were they responsible for the propagation of the worm onto the network. This action leads the SPAN Security Team to the conclusion 'hat the account had been compromised by an unknown individual. This conclusion is not too difficult to realize, since the password on the account was the same as the username.

The accounting records also show that on December 23, from 1:58-2:23 EST, the PHSOLIDE account was logged into again via the terminal server. Once logged on, this user read and deleted all the computer system banners from the nodes that returned this information to the 20957::PHSOLIDE account over the eight-hour period after the worm was released. Even though the actual banners had been deleted, the network transaction files revealed that 79 nodes sent their banners to the Switzerland computer. Of the 79, only 27 of these nodes were on SPAN.

Within an hour after the intruder collected the banners, then deleted them to cover his tracks, the DECnet line linking this computer to the rest of the DECnet Internet was disconnected for a scheduled maintenance. At this time the worm was still running on node 20957:: and continued to randomly select new nodes to propagate to. However, the only nodes available to this active worm were connected to the local area network at the university. During the next eight hours, of the 610 nodes on the local university network, the worm executed on 46 computers 90 times, with 15 computer nodes executing multiple versions of the program.

CLEANUP ACTIVITIES

A follow-up investigation by the SPAN Security Team several days after the worm was released revealed that over three-quarters of the known nodes (79) that previously executed the worm still had TASK Object 0 accessible as before. If needed, TASK Object 0 performs an important function by easily allowing the sharing of peripherals in a local environment. It was obvious that the deletion of TASK Object 0 from the operating system was not a permanent solution to a potential security problem. Since then, the SPAN Security Team has provided these nodes with several alternatives from which to chose. These procedures are outlined in a new release of the SPAN Security Policy and Guidelines [3] document.

At the university in Switzerland where the worm was initially released, a report was written and distributed campus-wide to alert the systems managers of the security problems they needed to address. Below is a list of the things the systems manager of node 20597:: insisted would be done campus-wide in addition to their existing security procedures.

a) There would be no multi-user accounts
b) Passwords would be required for dial-in access (through modems)
c) There would be a restricted user list for dial-in access
d) Additional accounting information would be required for terminal server access
e) Certain Username/Password combinations would not be allowed
f) A secure solution for providing TASK Object 0 program functionality would be implemented

It is important to point out that, in addition to the above, the first and most important practice in providing a rudimentary level of computer security rests with users, by their choice of passwords. Strict password control should be of prime importance for everyone on a computer system and its associated networks. For SPAN nodes, a new software audit system is available that will provide the system manager with tools to rapidly identify many other security weaknesses in the system in addition to the ones described above (send mail to NCF::Sisson for further details).

CONCLUSIONS

The Father Christmas Worm has over 150 lines of non-trivial control language code demonstrating a reasonable understanding of VAX/VMS and the DECnet protocol implementation on DECnet networks. It is obvious from the analysis of this event that the individual who released the Father Christmas Worm realized what he or she was doing and carefully returned again to the compromised node to collect information (system banners) indicating the extent of the worm on the DECnet Internet. It is also obvious that the perpetrator expected a large number of computers to receive and execute the worm, since the worm was released during the Christmas holiday season when there would have been the best chance of a worm executing on unattended VAX machines. In addition, it is typically held by computer hacker groups who make a habit of compromising the integrity of computer systems that computer systems managers, in general, do not implement appropriate security procedures and, therefore, are asking for unauthorized access to occur.

It is estimated that half of the 12,000 DECnet Internet nodes received the worm, but much less than 2 percent of those computers executed HI.COM within the first eight hours after the release of the worm. Within minutes after the worm was released, a very quick user reaction across the DECnet internet occurred, and the situation was immediately taken seriously. Once the source program for the worm was captured, a procedural cure, using existing functionality of the computer operating systems, was quickly devised and distributed by several

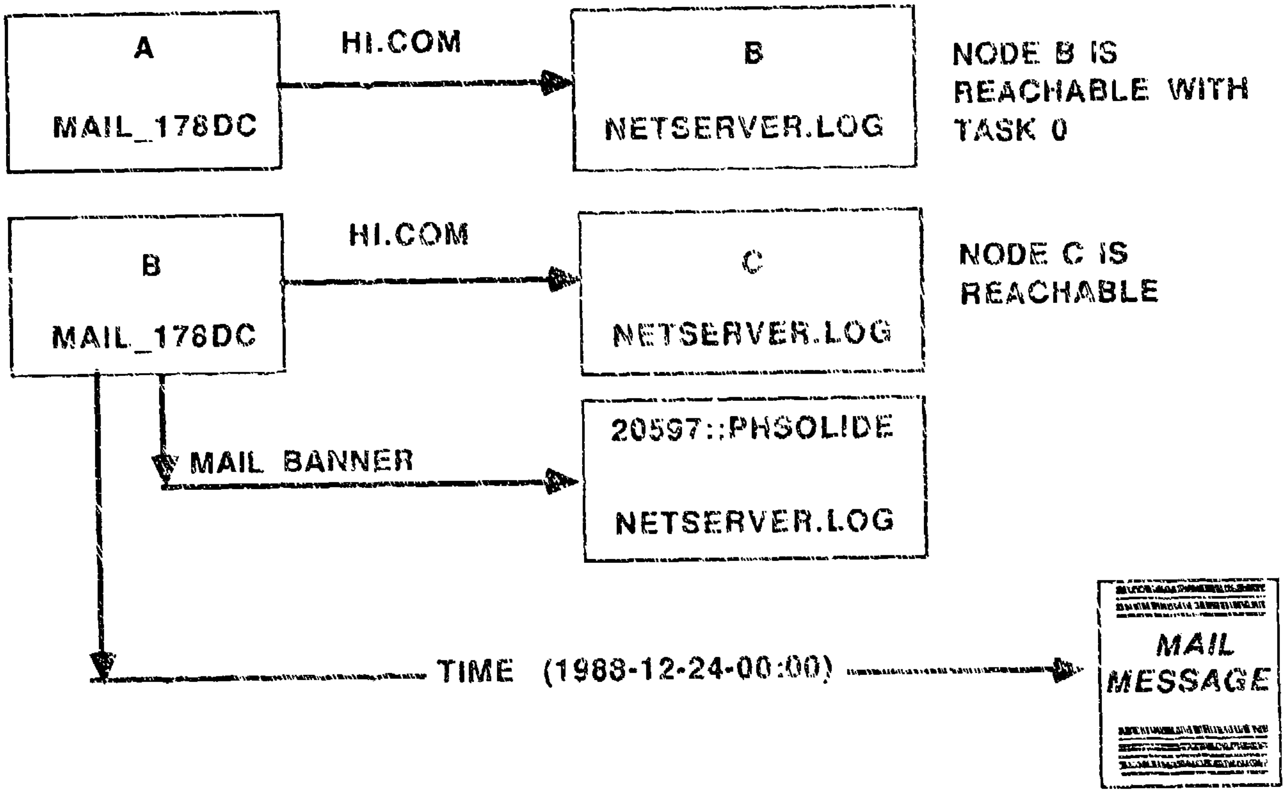


Figure 1: An overview of the major processes of the "Father Christmas Worm." In this example, Nodes A and B are executing the worm program HI.COM. Although Node C has a copy of the worm, it does not execute the program nor does it participate in the propagation of the worm because it has implemented certain security measures.

organizations that use the network. A combination of existing computer security measures, the quick and accurate procedures devised to stop the worm from executing, and the network itself were the main reasons the worm executed on such a small percentage of nodes.

On Friday, January 13, 1989, a worm nearly identical to the Father Christmas Worm entered the DEC internal network, called Easynet. The private Easynet network contains more nodes than the DECnet Internet. However, as discussed in a recent issue of *Digital News* [4], according to DEC the worm was spotted as it entered the network, and the system manager "was able to segregate the infected system before the worm could spread." It is believed that this incident was quickly controlled because of the widespread exposure and experience gained the previous month with the Father Christmas Worm.

Overall, the impact of the Father Christmas Worm was minimal in an operational sense but extensive in the area of strengthening computer system security (an ongoing activity). A process has been started to formalize procedures that will deal with worms, viruses, and other violations that threaten the DECnet Internet in the future. Key security personnel have been identified from each of the major networks in the DECnet Internet, and their responsibilities are being delineated.

Whatever may be the intention of the authors of computer worms and viruses, if these threats are not met head on and dealt with rapidly, the ultimate result may be that they destroy the productive working environment that an open network provides.

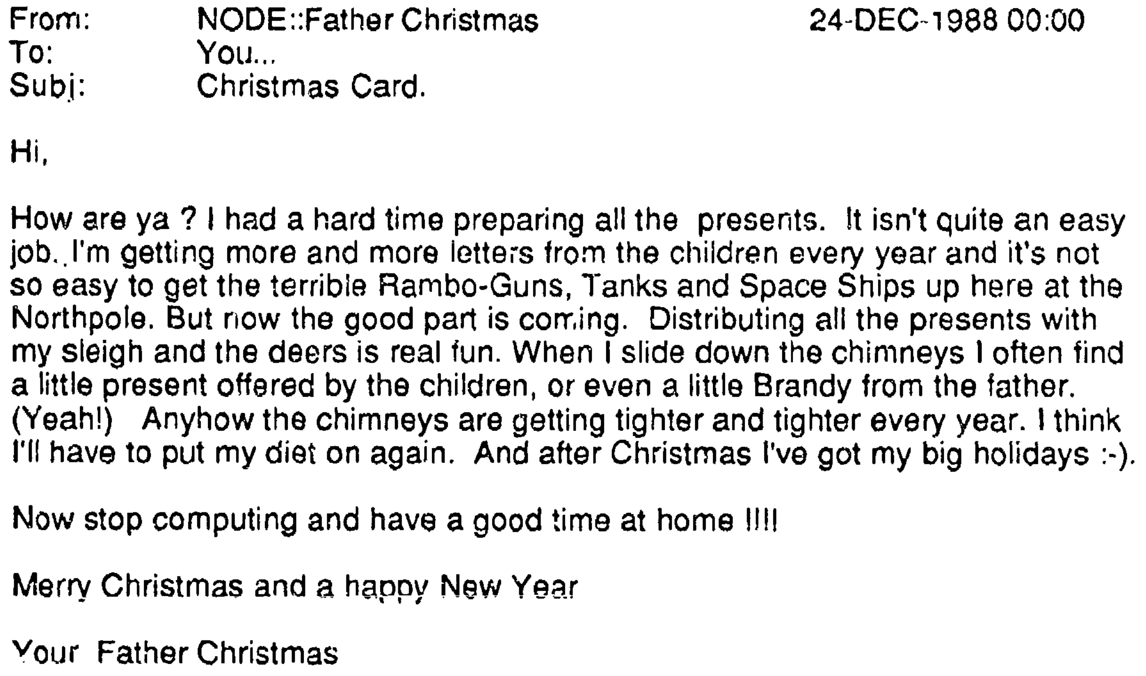
REFERENCES

[1] J. L. Green, V. L. Thomas, B. Lopez-Swafford, and L.Z. Porter, Introduction to the Space Physics Analysis Network (SPAN), Second Edition, NSSDC Technical Report, January 1987.

[2] V. L. Thomas and J. L. Green, "SPAN - A revolutionary tool for scientific research," Journal of the National Technical Association, p. 45, Winter 1989.

[3] P. Sisson, T. Butler, D. Peters, V. Thomas, and J. Green, SPAN Security Policy and Guidelines, NSSDC Technical Report, July 1989.

[4] S. Lawson, "Catching the worm," News Briefs, Digital News, January 23, 1989.



From: NODE::Father Christmas 24-DEC-1988 00:00
To: You...
Subj: Christmas Card.

Hi,

How are ya ? I had a hard time preparing all the presents. It isn't quite an easy job..I'm getting more and more letters from the children every year and it's not so easy to get the terrible Rambo-Guns, Tanks and Space Ships up here at the Northpole. But now the good part is coming. Distributing all the presents with my sleigh and the deers is real fun. When I slide down the chimneys I often find a little present offered by the children, or even a little Brandy from the father. (Yeah!) Anyhow the chimneys are getting tighter and tighter every year. I think I'll have to put my diet on again. And after Christmas I've got my big holidays :-).

Now stop computing and have a good time at home !!!!

Merry Christmas and a happy New Year

Your Father Christmas

Figure 2: The "Father Christmas Worm" electronic mail greeting. This message would only be sent to the users on a system executing the worm if it remained undetected until December 24, Christmas eve. After this mail message was sent, the worm program would stop executing.

An Epidemiology of Viruses & Network Worms

Cliff Stoll
Smithsonian Astrophysical Observatory
Harvard - Smithsonian Center for Astrophysics
60 Garden Street
Cambridge, MA 02138
617/495-7157
Cliff@cfa.harvard.edu

12th National Computer Security Conf.
Baltimore October 12, 1989

Abstract

By comparing worms that propagate over the networks, we can learn about the threats to our computing communities. These worms take advantage of operating system features as well as holes. They provide an adversary with both a denial of service weapon, as well as a means of gathering information. They can be studied with techniques developed for medical epidemics.

Introduction

In the past year, we've noticed several network security problems. What can we learn from these? How common are they? How many systems can an attack disable? How have people responded to these problems? Is our only worry the denial of service? How vulnerable are our networks? This paper addresses these questions.

IBM Christmas Tree Exec

On December 18, 1987 [4, 10], a program infected the IBM internal network. The software itself was disguised as an electronic mail message, under the name of "Christma Exec". In fact, it was an executable command script, which, when executed by a user, mailed copies of itself to others on the user's mailing list.

This took advantage of an operating system feature: the ability to execute a command shell script which has been received in the mail. The header line instructed the recipient not to unpack the program, but rather to execute the mail message immediately. It relied upon manual execution to replicate itself.

Although called a virus, this program was a manually propagated worm: a program which is copied from one network node to another. Indeed, since the program relied upon the gullibility of users, Bill Rubin of IBM Watson Research Center, considers it a trojan horse [10].

Within a few hours, it had entered many hundreds of IBM mainframe computers around the world. The load upon the individual systems was sufficient to disable many computers until they were re-booted. On a large mainframe, this can take an afternoon.

The program first arrived over Bitnet on December 9, 1987. The shell script instructed the user to execute it to receive a graphic of a Christmas tree on the screen. The first two screens of data were a drawing of a Christmas tree followed by a message, "Browsing this file is no fun at all, just type Christmas from CMS".

But when a user executed that command, the program searched through a user's nicknames files (VM "Names" files), and mailed a copy of itself to each user mentioned in that file. It did not erase itself after mailing itself away -- if it had, it would have been more difficult to track down.

Propagation speed through two different networks

The Christmas Exec worm was first found on the Bitnet network. Within a day, warnings were sent out to Bitnet sites. Gateways to Europe purged copies and thought they had it under control. Unfortunately, a day or two later, the program reached the IBM internal network, VNET.

The VNET outbreak was much worse than on Bitnet. Although Bitnet covers many more sites, only a fraction of them use IBM hardware; all nodes on VNET are IBM sites. Then too, VNET network is much faster -- 56 KBaud, as opposed to Bitnet's 9.6 Kbaud. The program could spread faster there. VNET users often had large Names files, which promoted the worm's spread.

The Bitnet infection occurred early in the day -- systems managers could react during the day to stop it. The Vnet infection, occurring late in the day, occurred while nobody was watching. Finally, the VNET network, as an internal network, is an internal network, and probably was trusted more than a public network like Bitnet.

This Christma Exec was the first network-propagated security problem -- the predecessor to network worms. Worms are not the same as viruses. A computer virus copies itself into another program, and lies dormant until the infected program is executed. A virus cannot execute alone -- it must be linked with a program. A worm, however, is a valid stand-alone program. It copies itself from one computer to another, usually over a network. Strictly speaking, a worm does not infect a disk copy of a program -- it executes within a computer, and when the executing copy is stopped, the computer is clean, unless re-infected from outside.

Of course, this taxonomy is simplistic. A malicious program may have sections of code which are worm-like or virus-like. And the Christmas Exec used a trojan horse to invite users to execute it.

November Internet Worm

On November 2, 1988, a self-duplicating program was released into computers attached to the Arpanet. This program has since been disassembled by several groups [1, 2, 3] and well described in the June 1989 Communications of the ACM[1].

Several salient points about November's Internet Worm:

1) It used multiple attack mechanisms, taking advantage of:
   a) two bugs in network interfaces
   b) common passwords
   c) entries in trusted hosts tables

2) It duplicated itself without manual intervention -- it was the first autonomous network worm.

3) The worm was tailored to infect both Sun and Vax computers, but could only run under the Unix operating system.

4) It was written to evade detection and understanding:
   a) it erased its argument lists
   b) it deleted the executing binary
   c) strings and constants were hidden by a hex mask

5) Within it, but unexecuted, was code to send messages to another networked computer.

This fifth point is important: as we shall see, other worms have been modelled after this one, and sent messages to central collection points as each new host is infected.

[1] These papers are models of workmanship and rigor; yet the authors are academics without support to study computer security.

How many systems were infected?

Although the microbiology of the November worm is now well understood, there have been no published epidemiologies describing the extent of the virus's spread. Reports of 6000 infected computers are based on only a guess.

To determine the extent of the worm's spread, I posted notices on the Internet, requesting anecdotal reports from both infected and non-infected sites. To insure widespread distribution of my requests, I posted these requests to several Internet forums, including Risks, Virus-L and TCP-IP forums.

The response to these requests was heartening. I received about 200 reports, of which 75 sites reported infected systems. Because of a lack of standardized reporting procedure, each report had to be separately analyzed to determine:

How many computers were infected?
How long were the systems disabled?
When was the infection discovered?
Why were some non-immune computers uninfected?

Most of the positive reports described multiple infections: a single person managed clusters of workstations. When the worm disabled one file-server, it disabled other workstations that depended on that server. The anecdotal reports often did not differentiate between individual computers being infected and multiple computers being disabled.

In October 1988, the Network Information Center's Domain Walking program counted about 60,000 computers attached to the Internet. What percentage of these were infected?

The 75 positive reports cited about 300 infected computers. Since many sites did not report, these reports alone do not determine how many systems were actually hit. However, several sites sent detailed computer generated logs, listing not only which computers attempted to infect their system, but also the times of each connection.

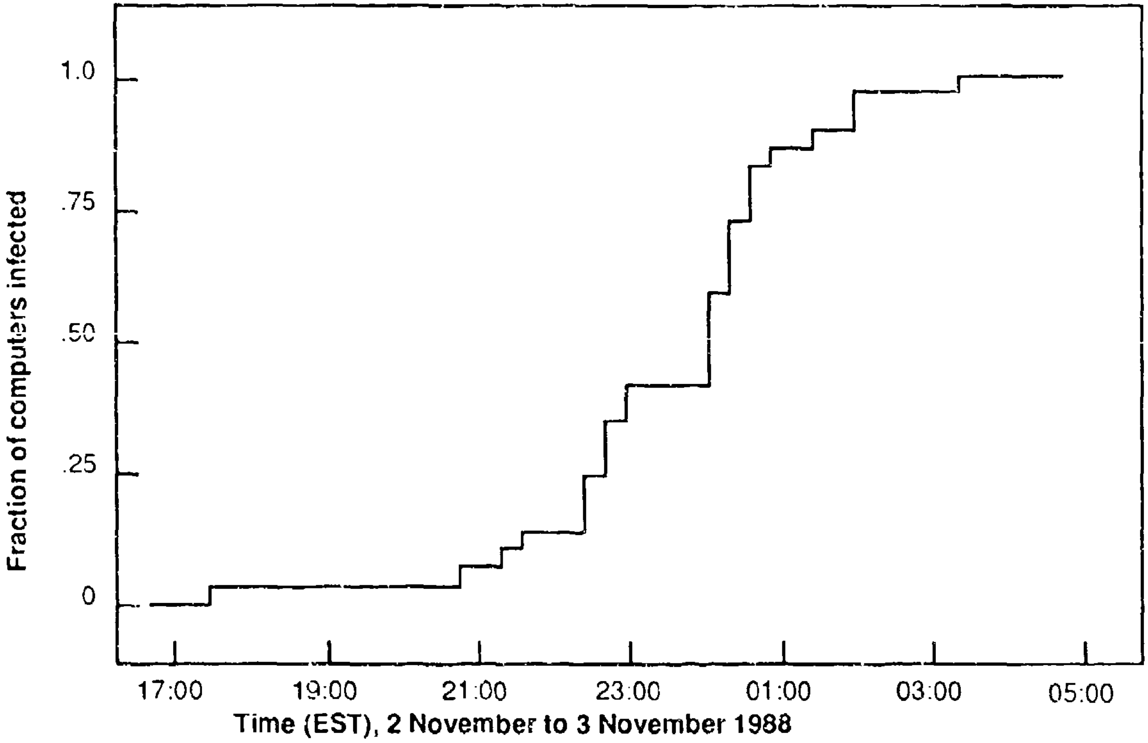
These two sets of reports (the human generated ones and the computer logs) are statistically independent measures of the same population. Indeed, some systems show up in both sets. By analyzing the cross-correlation between the two measures[7], we can estimate the total number of infected computers. We find that the worm entered about 2600 computers, with a 1-sigma error of 275.

This is a useful quantity -- and contrasts with media reports of 6000 infected computers. This initial report, estimated by Schiller at MIT [1] was only an informal estimate, and was not based on a detailed sampling of the thousands of computers on the Internet. Indeed, one of the most noticeable effects of the November worm was the loss of electronic communications, as managers isolated their systems.

2600 infected computers corresponds to about 4% of the total Internet population. Past studies [18] show a similar rate of insecurity for networked computers.

How fast did the Internet worm spread?

There's other information in the field reports we gathered. By combining the times of first infection, we can graph the number of infected computers as a function of time:



The shape of this curve is important: the rising part of the S curve corresponds to an exponential growth, as would be expected were the program limited only by replication time. In this section, the slope of the curve indicates the e-folding time of the worm. As the curve flattens off, we see growth limited by available systems. A similar shape would be expected in a biological population exposed to a contageous virus [16,17].

With few exceptions, most systems were unavailable for use while infected; this shows an amazing ability to deny service to a wide expanse of users.

Worm of December 1988

On December 23, 1988, a worm was spread in the NASA/SPAN - DOE/HEPNET networks [5]. These networks are crosslinked, and rely upon the Decnet protocol. Almost all systems are Vaxes running the VMS operating system -- a homogeneous population.

Previously, security problems have received wide publicity: In July 1987, the Chaos Computer Club in Germany reported that they invaded several hundred SPAN computers [6]. Unlike these previous attacks, which were manual, the December Worm automatically attacked individual computers on the SPAN network.

This worm, written as a VMS shell script, was not encrypted or obscured -- its techniques were immediately apparent. It would enter a computer through the Decnet Task object -- the software interface which lets outsiders run tasks on the computer.

A VAX/VMS system manager can disable the network Task object, but the operating systems were distributed with this object enabled. Probably a third of the computers attached to the network had this object enabled, and thus were vulnerable.

On a VAX/VMS computer, the task object runs non-privileged programs; in short, it allows a networked outsider minimal access to a system. Such a port would seem to be a "safe" option, since you cannot delete someone else's files from a job running through this interface. System administrators probably felt that the minor risk of this option was well worth the convenience to users.

The December Decnet worm was copied into a non-privileged account from an outside, networked computer. Once it entered a computer, this worm copied the system greeting/banner page, and mailed it to a computer in France. The worm then mailed a greeting to every user on the system, and then attempted to randomly infect other computers on the network. Each attack took a couple minutes -- within twelve hours, several hundred computers had been infected.

This worm implemented what the November Internet worm only hinted at: it mailed information to a central collection site. Whoever was at that computer could determine which systems were infected and (from the greeting page) what was happening at each site.

Within a month, another worm was launched on a different, private network. Remarkably similar to the December Decnet worm, this one searched for any accounts which had guessable or crackable passwords. Whenever such an account was discovered, the worm mailed the system name, account name, and password to two collection points, in distant parts of the world.

From this, we see that network worms pose dangers beyond simply denial of service. They can be efficient collectors of sensitive information. Even from unprivileged accounts they can steal information and send it to foreign systems, without knowledge of the system managers or users.

Analysis

What's common to these network worms? Each caused embarrassment to the network administrators, even though some might argue that no damage was done. Two of the worms (Internet and IBM) struck enough computers to effectively disable the networked computers.

Each worm propagated through existing network interfaces. Minor security problems in networking software and protocols can be exploited by worm writers [11]. Some worm writers exploit features intended to make life easier; for example, the finger daemon and Decnet task objects. Alas, future networking software should give strangers less help and privileges.

Even the worry of a worm attack can disable computers. On April 1, 1988, rumors spread of a logic bomb in Sun workstations. Again, on February 14th, 1989, unfounded rumors were heard about a malicious Valentine's Day greeting.

Diversity is important. Networks which have a single type of operating system are much more vulnerable than heterogenious networks. Bureaucracies will forever urge a single, standardized computing system, yet a diversity of operating systems insures survival against viruses and worms. Universally adopting any one standard -- Unix, VMS, TCP/IP or OSI -- will only make worms more destructive.

Can Worms be Good?

Workers at Xerox PARC [15] have developed ways to use distribute updates to databases using techniques similar to network worms. Suppose you have many small networked computers, each with an identical database. We need to update each of these database every week or so, say with new prices or stock information.

A central computer could call into each computer, and update each database. Alternatively, each computer could send the new database to a nearby networked node, after making sure that the nearby computer has not yet been updated. The new data spreads through the network as an epidemic. Such a database updating scheme is akin to a network worm; the developers used epidemiolgical techniques [16, 17] to develop these algorithms.

These epidemic algorithms are important developments in the use of networks and distributed databases. We can expect to see them commercialized in the next few years.

However, such database updating techniques are a far cry from the malicious worms described here. The protocols are agreed upon in advance, the software is designed for the purpose, and the network traffic load is small. They are "invited" into only limited numbers of computers, and designed to ignore other computers. Research into epidemic database techniques does not require experimenting with network worms or viruses.

Directions for future work

Much computer security research is directed towards securing isolated, multi-user computers[13]. Security problems, however, seem to show up on networked computers [12, 14]. In contrast to the Orange book, computers -- especially personal computers and workstations -- are often used by a single person. Communication is through networks, rather than mediated through an operating sytem. The very model used to write the Orange book is inappropriate and dated.

Viruses tend to be seen on personal computers, although Cohen's original experiments were on mainframe Vaxes. Worms can run only on networked systems, which today only extensively link large computers together. In the future, we can expect personal computers to be more widely networked, opening them up to such infections. Equally worrisome: financial markets, such as stock exchanges and commodities exchange markets, are being opened to networks [9], as are telephone systems.

Providing security in these environments is challenging. Simple user authentication -- whether by password, passcard, or biometric -- is inadequate. Programs themselves are difficult to assess for logic bombs and viruses.

We need ways to certify programs against tampering. Some methods to prove that a program has not been tampered with include embedded checksums and cryptographic certification. How can I distribute software, with each user certain that it has not been infected? Future research must address these questions.

For too long, computer security has been directed towards the creation of high-security, bulletproof systems. Real world computers are compromises in many ways; security is but one of these. We must find non-intrusive ways to allow the power of networking while maintaining the integrity of each computer.

References

1. Eichin, M.W. and Rochlis, J.A., "With Microscope and Tweezers: An analysis of the Internet Virus of November 1988". CACM 32, June, 1989.

2. Seeley, D. "A Tour of the Worm" in Usenix Winter 1989 Conference Proceedings, San Diego, Page 287.

3. Spafford, E., "The Internet Worm Program: An Analysis" Computer Communication Review Vol. 19, page 1, January 1989.

4. Internet Risks Forum, 7-22 IBM Christmas Tree Virus, December 1987.

5. LA Times "NASA Computers Infected" 26 December 1989.

6. Schmemann, S. "West German Computer Hobbyists Rummage NASA Files", NY Times Sept 16, 1987.

7. Bevington, Philip R. Data reduction and error analysis for the physical sciences, New York: McGraw-Hill, 1969.

8. Markoff, J. "Computer Snarl: A Back Door Ajar" NY Times Nov. 7, 1988.

9. Wall Street Journal Feb 3, 1989, "Chicago Commodities Exchange to Network".

10. Rubin, Bill. "Report from the Front - The Christma Exec", Proceedings Share '72 IBM Users Group Conference, Los Angeles, CA, Feb. 28, 1989.

11. Bellovin, S.M. "Security Problems in the TCP/IP Protocol Suite", Computer Communication Review, Vol. 19, page 32, April 1989.

12. Stoll, C. "Stalking the Wily Hacker", CACM Vol. 31, pg 484, May 1988.

13. National Computer Security Center. Orange Book, CSC-STD-001-83

14. Stoll, C. The Cuckoo's Egg, Doubleday, NY. 1989.

15. Demers, Gealy, Greene, et. al. "Epidemic Algorithms for Replicated Databse Maintenance", Xerox Palo Alto Research Center, February 7, 1989 (Also in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, August 1987)

16. Bailey, N. "The Mathematical Theory of Infectious Diseases," 1975.

17. Fraunethal, J. Mathematcal Modeling in Epidemiology, 1980.

18. Stoll, C. How Secure are Computers in the USA, Computers and Security, Jan. 1989.

An Assured Pipeline Integrity Scheme for Virus Protection

John Page
Mail Stop 5S3
Planning Research Corporation
1500 Planning Research Drive
McLean, VA 22102

ABSTRACT

Computer viruses pose a serious threat to the integrity of modern computer systems. Current approaches to securing systems do not address the particular dangers of the virus, particularly its ability to reproduce. Current integrity policies are analyzed with respect to computer viruses, and important requirements for a virus protection scheme are isolated. A virus protection integrity scheme based on the work of Boebert and Kain is presented that limits the virus's ability to reproduce and enforces virus protection at all stages of the software development process. This scheme is then applied to the software development process, and conclusions are drawn as to its effectiveness.

1 INTRODUCTION

Although there is no consensus as to the precise definition of a "virus," it is obvious that the computer virus and other strains of malicious software present a threat to secure systems that has no counterpart in the paper world. Combating the computer virus will require more than a simple patch to accepted practices, but rather a realization that overly simplistic notions of data integrity are not sufficient to regulate the behavior of executing code. What is needed, Cohen states [1], is a new awareness of data integrity on theoretical, practical, and social levels. Such an awareness would result in attempts to model system integrity in its own terms, rather than constraining it to mirror the terminology of data secrecy. This paper furnishes an integrity scheme to meet Cohen's challenge, extending the concept of the "assured pipeline," developed in [2], to prevent a virus from reproducing while restricting the behavior of other forms of malicious software. This paper will attempt to synthesize many current trends in data integrity into a unified integrity scheme that will protect software at all stages of the software development process.

2 THE VIRUS THREAT

The threat posed by the computer virus is a multifaceted one, particularly because the distinction between viruses and other forms of malicious software becomes blurred in practice. Much of this confusion stems from the unclear notion of "infection." It has been argued that a virus is a hostile piece of code in an executable that propagates itself by prepending itself to other executables when activated [3]. This definition, however, would not account for source code viruses. Another interpretation, [4], states that a virus need not "infect" any one program, but may, more abstractly, "infect" a host system by multiplying within it and feeding off of its resources. We may also talk of a virus "infecting" a network. In each case, the granularity of "infection" is drastically different. Unfortunately, the creators of viruses are not constrained by our definitions, so we are forced to look for more general solutions to the malicious software problem.

For the sake of argument, let us start with a rather narrow definition of a virus, a definition which will be broadened in the course of this paper. We can define a virus as a hostile segment of code in an executable that, when executed, attempts to prepend itself to other executables. Such a virus, by this definition, would be a form of Trojan horse in that it lies dormant until its host is activated and, when activated, it usurps the authorization of an unwitting user. In addition, this virus may be presumed to contain some form of logic bomb, programmed to launch an attack upon the system in response to a predetermined stimulus. This attack may be a denial of service, whose prevention remains an intractable problem. While these threats have already been recognized, the virus has an additional strength in its ability to reproduce. This ability gives the virus an unprecedented degree of mobility as well as the strength of numbers, which can be used as a means of attack (e.g., crashing a network through unrestrained multiplication). The numerical advantage also serves as a defense mechanism for the virus code. Of course, these attributes are also attributed to "worm" programs, further complicating the distinction of what constitutes a "true virus."

It is important to realize that the virus, no matter what definition we choose, is a threat to the integrity of a system, not the secrecy of any data that the system may contain. When a virus reproduces, it modifies the objects of a system as opposed to observing them. As a result, a system which is secure with respect to secrecy alone does not offer any significant protection against an invading virus, since secrecy labels in such a system reflect the gravity of observing a labeled object, as opposed to modifying it. Cohen [5] notes the following paradox which occurs when one attempts to protect a given file under such a system. Since the system would have to enforce some notion of the "no write down" rule, the system would disallow any attempts to write from a high secrecy level to a lower one. Although this would prevent a program executing at a higher level from infecting a lower one, the reverse would be allowed. Therefore, the only way to protect a critical program from illegal modification is to label it lower than any subject or object in the system!

3 VIRUSES AND INTEGRITY MODELING

It follows from the preceding argument that any security model attempting to provide virus protection must support a strong data integrity policy that restricts the modification of executables. Before presenting the framework for such a policy, let us first survey some of the relevant issues in integrity modeling. The current state of the art in integrity modeling is currently at a crossroads. The pioneering efforts of Biba [6] (primarily his strict integrity policy model) enforced integrity via the same lattice structure used in secrecy modeling. Recent authors, however, have argued that problems in implementation motivate other approaches to integrity modeling [2] , [7]. Two models to date have provided an integrity policy specifically for virus protection. In this section, we will survey the integrity issues relevant to computer viruses and isolate the requirements for an effective virus integrity scheme.

3.1 User Integrity vs Program Integrity

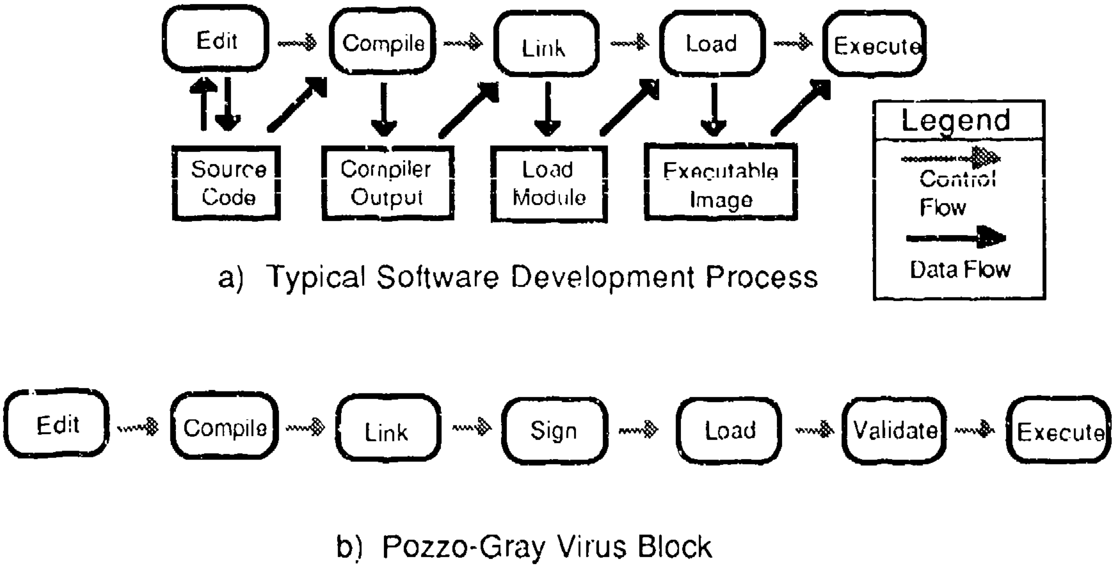
Should integrity be enforced in terms of users or programs? Biba [6] advocated assigning subject integrity labels to human users. This was motivated by his concern about the "human threat" to data integrity, where corrupt data could be introduced into a system directly from a disgruntled user. With this threat in mind, we are concerned with the degree of trust associated with the user, and not with the possibility that an integrity attack is being launched by the software the user is running. Therefore, in the formulation of his integrity policies, Biba chose not to address the internal threat represented by a computer virus or other forms of malicious software.

In the context of a computer virus, a purely user-based integrity scheme becomes meaningless, since the user who executes a virus-infected program has no knowledge of its presence. It is the executing program that represents the integrity threat, not the user. For this reason, Boebert and Kain argue that integrity should be associated with programs rather than users [2]. As we shall see, their work offers a promising point of departure for the creation of a virus integrity policy.

Both approaches have their merits, since they respond to different threats. A robust integrity policy must account for malicious users as well as programs. Such an integrity policy may be found in the Clark-Wilson Integrity Model [7], which enforces integrity on the basis of the triple (<user>, <program>, <data item>). In short, a protected item may only be accessed by specified users in a specified manner. Although this model successfully combines both user and program integrity, the fineness of granularity (and corresponding cost in overhead) may not always be necessary. The virus integrity scheme offered in this paper incorporates both user- and program-integrity schemes with a somewhat coarser level of granularity.

3.2 Linkage Protection

Two security models offer a mechanism to prevent a virus from infecting executables: the Pozzo-Gray Virus Containment Model [3], and the Argus Security Model [8]. Although the models differ greatly in scope and intent, they both protect linked object code from unauthorized modification. The Pozzo-Gray model is not so much a formal model as it is a framework for implementing a virus block. It describes how a cryptographic checksum may be used to verify that an executable has not been modified after linkage. The role played by the checksum in a typical software development process is shown in Figure 1.[1]



a) Typical Software Development Process

b) Pozzo-Gray Virus Block

Figure 1: Pozzo-Gray Virus Block

The Argus model is a more abstract, general-purpose model. One of its most noteworthy features is its ability to model the execution of programs via processes explicitly. This feature allows accountability to be shared between both a user and the software he is executing. The model blocks the infection of executables by only allowing their modification during the execution of a specially designated "Linker" file. The model also defines the roles of a Security Watch Officer (SWO), who is empowered to alter the "Linker" status of a file. This safeguard prevents a virus from designating "Linkers" of its own.

[1] This figure is based on that presented in [3]. It is included since it provides a useful context for the content of section 5.

An important theme unites the two models. Since they both restrict the type of program or process which may modify executables, they both enforce a limited type-based (as opposed to hierarchical) integrity policy. This raises a crucial issue in virus protection: any viral integrity policy will need to recognize types independently of hierarchical labels. Both models, admittedly, share the same weaknesses. The first weakness lies in the unpleasant possibility that the linker itself is infected with a virus. Although the Argus model offers some safeguards in this respect, both models require us to "trust" the linker programs. A far more serious shortcoming exists in that both models observe a very narrow definition of a virus. Suppose an invading virus were to attack the source code, a shell program, or intermediate output of the compiler? Such a virus would pass through linkage protection mechanisms with ease. What is needed is an integrity schema which would offer protection at *all* stages of the software development process. This protection may be realized by the adoption of an *assured pipeline*.

3.3 The Assured Pipeline

An assured pipeline is a subsystem divided into chronological stages. In each stage, the types of objects that may be observed or modified is strictly limited. In addition, a process "entering the pipeline" is constrained to traverse it in a pre-defined order. The concept of the assured pipeline forms the central tenet of Boebert and Kain's groundbreaking work on integrity [2], and forms the basis of the Type Enforcement mechanisms for the Honeywell LOCK prototype [9]. Its creation was driven by their attempts to secure the integrity of a module which labelled output data. They found that hierarchical labeling of the intermediate data was powerless to constrain the proper flow of data and execution control. (A sole reliance on hierarchical labels, in fact, would make it necessary to invoke a trusted subject at every stage of the process!) Instead, they proposed the idea of an assured pipeline based on the notion of *execution domains*. By restricting the activity in each domain, and by placing constraints on the transitions between domains, they were able to safeguard labelled data from being intercepted before it was output. The pipeline offers a powerful mechanism for establishing flow control. Even if a call to the pipeline is initiated by a hostile segment of code, the data and control flow within the pipeline is still regulated. This measure may help to ward off more sophisticated virus attacks which attempt to force a benign subroutine to perform the infection.

4 AN ASSURED PIPELINE INTEGRITY SCHEME FOR VIRUS PROTECTION

Any successful virus defense must address the preceding issues. Integrity enforcement must govern both users and programs. Both processes and the programs they execute must be modeled explicitly, as in the Argus model. Such a defense would need to employ a type-based integrity policy independently of any hierarchical one.[2] Finally, it must provide some measure of flow control to restrain the hostile invocation of benign code. A modified assured pipeline scheme will meet these requirements. The framework for such a scheme is presented in this section, in the form of a set of definitions and four properties.

This scheme is intended to be as model- and architecture-independent as possible. Therefore, the granularity of enforcement presented in the following two sections is arbitrary: readers are encouraged to experiment with different configurations. Any implementation would prove much less costly with some type of role or group strategy. This scheme recognizes the following definitions:

Users: "Human" users of a system.

[2] This is motivated by the inability of hierarchical labels to create an assured pipeline. Should there be a need for hierarchical integrity enforcement (perhaps in a DBMS application), it is assumed that this would be implemented separately. The reader is also advised to investigate the Risk Management policy described in [3] before completely abandoning the notion of hierarchical integrity enforcement.

Data Objects: An "object" in the traditional security modeling paradigm. In most cases it will only be necessary to enumerate the types of objects rather than specific instances of a given type.

Program: A routine, subroutine or program *in execution* by a process. The code representing the program (source, object, etc.) is considered as an appropriately typed data object.

Process: The "meta-program" in execution which executes "programs" on the behalf of a given user, maintaining the process control block for each executing program.

Integrity modes: The access modes of observe (**O**), modify (**M**) and observe/modify (**O/M**).

Domain: A set of pairs of the form (<data object>, <integrity mode>).

Execution Domain: A domain which is associated with a process in execution. Any program executing under that process may only observe or modify data objects as specified by that domain.

User Domain: The set of execution domains which may be entered by processes running on the behalf of a given user.

Execution Domain Table (EDT): The set of all execution domains in tabular form.

User Domain Table (UDT): The set of all user domains in tabular form. They may be defined on a per-user basis, or by user groups or roles.

Program Domain Set (PDS): A set of allowable execution domains associated with each program.

Execution Modes: Any members of the set (**Null**, **Run**, **Call**<domain>), where <domain> is one of the domains recognized by the system. The significance of these modes will be made clear in section 5.1.

Domain Transition Table (DDT): A two-dimensional array indexed by the set of execution domains, and whose entries are execution modes.

Before we go any further, let us reflect upon how these definitions interrelate. We start with the assumption that a process is executing somewhere within the system. This process is associated with a given user, and is currently running in a given execution domain. This process may or may not be running a specific program. We now define the basic properties needed to provide the proper integrity enforcement:

Process Integrity Property: A process executing within a given execution domain (or program executing in the context of that process) may only observe or modify a given data object if there is a corresponding entry in the Execution Domain Table for that domain and object.

Program Integrity Property: A program may only execute in a given execution domain if that domain is included in the program's Program Domain Set.

User Integrity Property: A process associated with a given user may only enter the execution domains specified by the User Domain Table.

The discussion so far has centered on processes executing within a given execution domain. What happens when a process attempts to move from one execution domain to another, or attempts to access an object outside of its domain? We assume that a process attempts a domain transition if it attempts to access an object in a manner not allowed by the current domain. In order to fulfil its task, the process (or program acting underneath it) needs to select an execution domain where the access would be allowed. The mechanics of this selection would depend on each particular implementation. The three static properties previously described must hold in the new domain. In addition, the following transition property must hold:

> Domain Transition Property: A transition between domains for a process is legal only if there is a corresponding entry in the Domain Transition Table.

If these conditions are satisfied, then the process in question can now be considered to be executing in the new domain.

This formulation addresses all of the integrity issues discussed in the previous section. It offers integrity enforcement on both the user and program levels like that of the Clark-Wilson model. It encourages the type-based integrity enforcement necessary to resist virus infections. It has the flow control benefits of the assured pipeline. Finally, it not only protects executables, but resists virus attacks at all stages of the software development process, as is shown in the next section.

5 THE SOFTWARE DEVELOPMENT PROCESS

We now apply this integrity scheme to the simplified software environment described in section 3. An application of this scheme is achieved by defining all of the necessary domains. The user interface can be modeled internally by the set of allowable transitions between execution domains. Each of these domains may then be described in terms of the objects they regulate. Sets of execution domains are then defined for all programs (or classes of programs) recognized by the system. Finally, the role of user domains in limiting viral activity can be demonstrated.

5.1 Domain Transitions

We start with a user's view of a simplified software development environment, mapping it onto a set of execution domains. A user, currently executing commands from the command line interface, wishes to edit source code, compile and link it, and , finally, load and execute it. We now express this in the form of execution domains and the allowable transitions between them.

Command: The command line interface. Transitions may be made directly to the Edit, the Compiler, and the Loader domains.

Edit: A text editor. Transitions may only occur between the editor and the Command domain.

Compiler: The process which takes source code as input and produces unlinked object code and temporary files as output. This domain may only be entered from the Command domain.

Linker: The process which takes the compiler output as input, and in turn outputs an executable load module. We add the additional restriction that the linker domain may only be reached from the compiler domain. This restriction will prevent a hostile process from attempting to link the object code with a malicious external subroutine.[3]

[3] This paper assumes the existence of some token-passing protocol (outside the scope of this paper) to allow the linking of separately compiled modules. An absence of such a protocol would force complete recompilation.

Loader: The process which copies the load module into the user's process space for execution. The Loader may only be reached from the Command domain.

Executor: The domain in which the newly created program is executed. We arbitrarily add the restriction that the Executor is only to be reached from the Loader, and not from the Command domain.

The resulting configuration is not exactly a "pipeline," but the more generalized graph structure represented in Figure 2, which portrays the user interface with the software development subsystem. The user may switch between the Command and Edit domains at his discretion, but neither he nor any software he is executing may move from the Edit domain to any other domain without first going through the Command Domain. The Compiler and Linker Domains form a "mini-pipeline." Any process entering the Compiler domain must pass through the Linker domain and then return to the Command domain. (It may be assumed, for argument's sake, that any compilation errors are reported to the user by the Linker.) Correspondingly, the Linker domain may only be entered from the Compiler Domain. A similar structure also exists for the Loader and Executor Domains. Any process entering this structure must traverse both domains before control returns to the Command Domain.

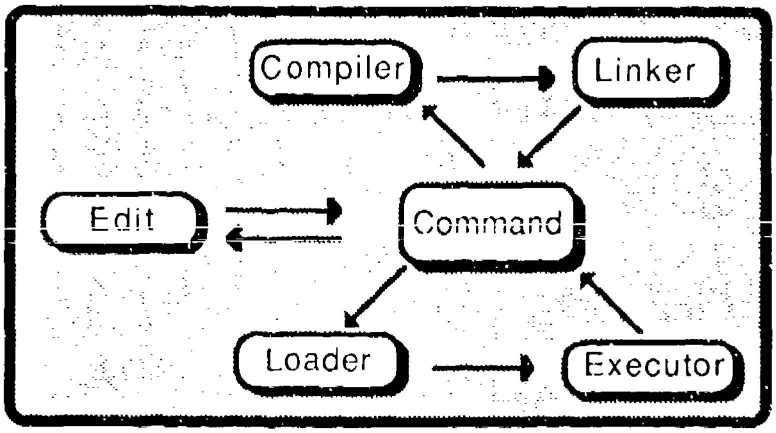


Figure 2: Domain Flow for Software Development Environment

The next step is to create a Domain Transition Table to model this configuration. If the process wishes to remain in the same domain as previously, **Run** is entered in the table.[4] If a domain change is attempted, we need to insert **Null** if the transition is not allowed, and a **Call** <domain> if it is allowed. The DTT for our example appears in Figure 3.

5.2 Data Objects and Programs

Now that the interaction between execution domains is clear, we need to examine each domain separately. This examination takes two forms. First, we need to define the data objects which can be accessed in each domain. Then we define the Program Domain Sets which prevent malicious code within a program from executing in unauthorized domains. We only discuss those data structures which

[4] An interesting problem arises when a malicious user or subroutine attempts to threaten the integrity of an item by repeatedly applying an authorized process against it. Neither the Boebert and Kain nor the Clark-Wilson approaches attempt to provide a solution. A token capability scheme such as that of Karger [10] seems to offer the best solution to such a problem. A temporal capacity may also be added to prevent a malicious program from infinitely cycling within a given domain, but this reaches beyond the scope of this paper.

concern the software development process in this section. In the next section, where user domains are discussed. we will examine the software development process in a much broader context.

Next Current	Command Domain	Edit Domain	Compiler Domain	Linker Domain	Loader Domain	Executor Domain
Command Domain	**Run**	**Call** Edit	**Call** Compiler	**Null**	**Call** Loader	**Null**
Edit Domain	**Call** Command	**Run**	**Null**	**Null**	**Null**	**Null**
Compiler Domain	**Null**	**Null**	**Run**	**Call** Linker	**Null**	**Null**
Linker Domain	**Call** Command	**Null**	**Null**	**Run**	**Null**	**Null**
Loader Domain	**Null**	**Null**	**Null**	**Null**	**Run**	**Call** Executor
Executor Domain	**Call** Command	**Null**	**Null**	**Null**	**Null**	**Run**

Figure 3: Domain Transition Table

We now define the following types of data objects, along with some restrictions on how they may be observed and modified:

Source Code: Text representation of a program. We stipulate that it may be both observed and modified in the Edit domain, and may also be observed in the Compiler domain. We also allow source code to be observed in the Command domain so that a source listing may be printed.

Compiler Output: Unlinked object code and any temporary files or structures produced by the compiler. Any data of this type may be modified in the Compiler domain and observed in the Linker domain.

Load Module: Linked object code which is ready to be copied into a user's process space in primary memory. It may only be modified (or created) in the Linker domain and observed in the Loader domain.

Executable Image: The image of the load module which has been copied into the user's process space. It may be modified in the Loader domain, and may be both observed and modified in the Executor domain. (This example allows self-modifying code. It may easily be forbidden by altering the EDT entry for the Executor Domain.)

User Objects: This class of data objects covers any data structures "belonging" to a general user that have not been enumerated above. We stipulate that these objects may be observed and modified in the Command domain or in the Executor (i.e., they may be accessed by running programs).

By correlating these data types with the set of execution domains, it is possible to generate an Execution Domain Table as shown in Figure 4.

We shall address the definition of Program Domain Sets in less detail, since their use is relatively straightforward. The PDS defined for a text editor, for instance, need only contain the Edit domain. This would not only prevent viral infections, but would limit any Trojan horse activity outside of the Edit domain. The Compiler and Linker may appear in the form of one program allowed to run in both domains, or two separate programs which run in isolated domains. An important issue in PDS definition arises when the transitions between domains are initiated. If a program (as opposed to a process) attempts to initiate a transition, it must have both the source and target domains in its PDS.

Type Domain	User Objects	Source Code	Compiler Output	Load Module	Executable Image
Command Domain	O/M	O	Null	Null	Null
Edit Domain	Null	O/M	Null	Null	Null
Compile Domain	Null	O	M	Null	Null
Linker Domain	Nuli	Null	O	M	Null
Loader Domain	Null	Null	Null	O	M
Executor Domain	O/M	Null	Null	Null	O/M

Figure 4: Execution Domain Table

5.3 User Domains

It is natural to assume that user-based integrity controls would not prove to be of any use in virus control, since the virus operates beyond the knowledge of the user. This assumption, however, is somewhat misleading. Although user-based integrity controls do not provide a sense of "absolute protection," in the proper environment they may prove to be a valuable weapon in virus defense. Consider a computer network for a large organization. While the majority of the network users run a limited library of applications, only a small fraction would be actively involved in software development and maintenance. Few users would be expected to modify executable code. A sensible user integrity policy could prevent infections from users not authorized to modify programs, thus narrowing the number of users who could spread a virus infection.

A typical organization with such a policy is summarized in Figure 5. This table specifies which domains may be entered for each user. Although all users will need to use the editor, only E. Poe is expected to compile and link program code. Since he is the only user expected to create or modify software, a virus triggered by other users will not be able to reproduce. (In this example, Mr. Poe is portrayed as a "superuser." It would be prudent in a serious implementation to divide Mr. Poe's omnipotence into several user roles.) The precise definition of domains causes some difficulties, however. Although all users should be allowed to run routines from the library, we wish to make sure that only Mr. Poe is allowed to modify these routines. The only way to model this is by creating separate domains for both Mr. Poe and the other users. The domain for Mr. Poe, "Library," allows him to both observe (i.e., "execute") and modify library routines. Other users, however, are limited to the "Library User" domain, which only allows the observation of library routines. This type of domain definition can be used to create and define user roles, constraining the programs which may be used by each role. Note, however, that a user-based integrity scheme will be of little use in a development environment where most users create and modify software.

User	Department	Allowable Domains
W. Irving	Word Processing	Command, Editor, Loader, Executor, Proposals, Library User, Personnel
N. Hawthorne	Accounting	Command, Editor, Loader, Executor, Payroll, Library User, Accounts Payable
E. Poe	Software Development	Command, Editor, Compiler, Linker, Loader, Executor, Library, System Library User
H. Melville	Word Processing	Command, Editor, Loader, Executor, Proposals, Library User, Personnel
E. Dickenson	Marketing	Command, Editor, Loader, Executor, Proposals, Library User, Personnel, Resumes, Briefings

Figure 5: Sample User Domain Table

6 COMMENTS AND CAVEATS

How effective would such a scheme prove against a virus attack? It is at least as effective as the Pozzo-Gray and Argus models in only allowing a linker to modify (or create) executables. The approach presented has two distinct advantages over these models. First, it can control the circumstances under which the linker is called. Second, it offers similar protection throughout the development process, instead of at one point. Unlike the Boebert and Kain schema, it factors in a degree of user accountability, and draws a distinction between programs and processes. The Program Domain Sets can regulate the behavior of a program, which not only limits the spread of viruses but also the types of attacks a virus may launch against a system. Even if the bulk of the program has been modified, the program is still restricted to its original set of domains.

Some cautionary words are in order. First, a system is only as secure as its labelling. The tables and domain sets must be protected in order for such an approach to prove feasible. Access to execution domains which would permit a process to modify domain tables and sets would need to be restricted as much as possible. However, the scheme provided here does make it possible to define a SWO role for a trusted user, and the domain tables and sets can be treated as any other form of data object.

Second, this approach assumes a somewhat "conventional" (albeit highly structured) environment. It would be of little use in a symbolic programming environment. This scheme as it is, does not address parallel architectures or interpreted programming environments. Finally, although it is influenced by the Honeywell LOCK (formally SAT) architecture[2], some interpretation would be needed to apply it to a specific architecture. The demands it places on configuration management will be quite high. The price is not too excessive, however, when one considers the alternative.

7 CONCLUSION

The problems raised by the computer virus and other forms of malicious software have highlighted the need to address integrity as something other than a poor relation to data secrecy. The integrity needs of a functioning computer system cannot be satisfied by modeling integrity after the paper world of classified documents, for documents in a safe do not multiply out of control, nor do they control the behavior and contents of other documents. A new paradigm is needed, one that explicitly recognizes the integrity needs of a computer system. It is hoped that the scheme offered here, a synthesis of many previous integrity modeling concepts, may serve as a stepping stone towards a more unified, comprehensive view of data integrity for computer systems.

REFERENCES

[1] Cohen, F. "On the Implications of Computer Viruses and Methods of Defense," Computers and Security, Vol 7, No. 2. Netherlands, 1988.

[2] Boebert, W. E. and Kain, R. Y., "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, Fort George G. Mead, MD, 1985.

[3] Pozzo, M. and Gray, T. "A Model for the Containment of Computer Viruses," Proceedings of the AIAA / ASIS / DODCI 2nd Aerospace Computer Security Conference, December 1986.

[4] Eichin, M., and Rochlis, J. "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," Proceedings: 1989 IEEE Symposium on Security and Privacy, Washington, D.C., 1989.

[5] Cohen, F. "Computer Viruses: Theory and Experiments," Proceedings of the 7th DOD/NBS Computer Security Conference, 1984.

[6] Biba, K. J., Integrity Considerations for Secure Computer Systems, ESD-TR-76-372, Mitre Corporation, Bedford, MA, 1977.

[7] Clark, D. D. and Wilson, D. R. "A Comparison of Commercial and Military Computer Security Policies," Proceedings: 1987 IEEE Symposium on Security and Privacy, Washington, D.C., 1987.

[8] Adkins, M. "The Argus Security Model," Proceedings of the 12th National Computer Security Conference, Fort George Meade, MD, 1989.

[9] Boebert, W. E. "Constructing An INFOSEC System Using LOCK Technology," The Lock Demonstration, Distributed at the 11th National Computer Security Conference 1988.

[10] Karger, P. "Implementing Commercial Data Integrity with Secure Capabilities," Proceedings: 1988 IEEE Symposium on Security and Privacy, Washington D.C., 1988.

COMPUTER CRIME AND ESPIONAGE: SIMILARITIES AND LESSONS LEARNED

Lloyd F. Reese
Department of Veterans Affairs
810 Vermont Avenue N.W.
Washington, D.C. 20420

The insider threat (both accidental and intentional actions) continues to be the greatest threat to computer systems. This paper will address only intentional actions which probably constitute as great a threat to computer systems as fire.

When making presentations, I often start with asking a question. Is computer security a technical problem or a people problem? I usually get answers indicating that it could be either a people problem or some combination of both. With computer crime and espionage, it's more likely that I would get "it's a people problem" for a response. We now have identified the first similarity between the two, i.e., they both require people.

What about other similarities? They both involve some negative impact on an organization. In the case of computer crime, it may be the loss of funds, other resources, data integrity, or denial of service. In the case of espionage, the loss is information which may be quite crucial to national security. In both cases, a person or persons subverted the controls that were supposed to protect something of value.

In espionage cases the classic explanation for motivation is summed up by the acronym MICE: money, ideology, compromise, and ego. In recent espionage cases, little money was received by the perpetrators other than the Walker-Whitworth case which may have involved the payment of $1.5 million over several years. This case was primarily an ego trip for John Walker. The money was just a way to keep score [2,101].

Lonnie Moore at the DOE Lawrence Livermore National Laboratory conducted a study of recent espionage cases which he presented at the 1988 DOE Computer Security Conference. Moore found that the typical perpetrator was a male, about 39 years old, in a low status job with lots of responsibility and access to sensitive information. Greed was a primary motive in some cases, but relationships with a lover, spouse, or friend were a critical factor in one-third of the cases. Ego is a major issue: a person with a bruised ego represents a threat. Ideology was seldom a factor and compromise was unimportant. In Moore's research, the important factors were ego and relationships, with money a distant third [5].

In computer crime cases, money is certainly a motivator. The U.S. Department of Health and Human Services report of 1984-85 involved interviews with 46 perpetrators of fraud. The study attempted to understand why the perpetrators got involved with fraudulent activity. They were employees with federal, state, and local government or private agencies administering federal programs. They were young, good employees, and most had above-average performance. Only 20 percent had prior criminal records. Most were in positions where they could cause checks to be issued. The average loss was $45,000, but 20% were over $100,000. Seventy-five percent say they stole money in response to situational stress (7,i). One-third of this group indicated that they were also unhappy employees and that made it easier to commit the crime [7,11-12].

The report suggested debriefing perpetrators, better personnel security procedures, and improving system controls and awareness, but did not suggest employee assistance programs or training managers. By contrast, Moore suggested "crisis intervention" to help an individual with a problem. He also suggests the need for management to be good and fair [5].

From the preceding discussion, we know that people are tempted by money, influenced by relationships and situational stress in their personal lives, as well as unhappiness at work. To develop a countervailing program, we must next look at what we can and cannot control.

Clearly, we must have systems that process funds, and store sensitive or classified information. However, we can make sure that proper controls are in place and that they work. By contrast, we can do little about an individual's relationships. We can tell employees to exercise care in contact with foreign nationals or individuals with questionable backgrounds and intentions. Yet, it is quite difficult to control employees' associates after the workday has ended. We can help people deal with situational stress in their personal lives and we can also encourage good and fair management practices as well as other environmental factors that contribute to a pleasant work environment.

Let's look at situational stress. Have any of us every had any of the following happen: divorce, death of parent/spouse/child, illness of parent/spouse/child, financial difficulties, or alcohol/drug dependency. These life events that can place considerable stress on anyone. How do we react to these situations? That varies considerably from one person to another Some of us handle each within the usual norm. Some of us deal with them more quickly and some take longer. Sometimes we work out things by ourselves and sometimes we need help. When it takes longer than usual and a situation requires help, do we get it? If so how? Do we do so on our own or only with the helpful intervention of family, friends, or work associates?

It is important that supervisors and managers stay in touch with what's going on in the lives of their employees and be especially observant if they know the employee is having a difficult time in his/her personal life. When that personal difficultly impacts job performance, the supervisor can intervene and insist that the employee get help. Long before it becomes a serious on the job problem, the supervisor can suggest to an employee that certain services are available if the employee thinks they would be useful. When job performance is impacted, the supervisor can insist that the employee consult with the Employee Assistance Program (EAP).

The question then arises, how much confidence do we have in such programs? What type of credibility do they have with employees? I was concerned about the EAP program at my office. I decided to consult with the doctor at the health unit who serves as the initial counselor and referral for most employees with problems. I found that she and the nurses were running a mini-crisis center. I was impressed! However, some EAP programs are associated with drug and alcohol abuse only; others may be seen as for blue collar employees only.

It's important to learn how well your program is working. Can it serve as a place for the employee with situation stress to obtain help so he/she will not allow that stress to have a significant impact on the organization?

In addition to programs that deal with individual problems, it is also important for organizations to review internal environmental factors that can contribute to computer crime. What are some internal environmental factors that can contribute to computer crime? These include the work environment, reward systems, level of interpersonal trust, level of ethics, level of stress (pressure for performance), and level of internal controls. While organizations spend considerable attention on internal and accounting controls as well as defensive measures such as physical security, considerable less thought is give to the enhancement of the work environment, the reward system, levels of trust, ethics, and stress. There is no question that the latter factors are more difficult to assess as risks, but it is dangerous to ignore them [3,25-30].

How do most employees form their opinion of whether or not an organization is a good place to work? For the most part they base this conclusion on what they experience and what they hear from other employees. Clearly, what they see and hear from management is a significant influencing factor. If management is caring and operates in an open environment, the employees are more likely to have a positive feeling about the organization and less likely to want to take actions against it. If on the other hand, management is less benevolent, employees may be more likely to take advantage of the organization.

What can we do if the internal environment of our organization is less than ideal? Probably not a lot, but it's worth trying to improve it by working with management and employee organizations. Many of you have used self-assessment questionnaires to determine the status of security and controls in your organization. You can encourage the use of self-assessment questionnaires to measure organizational climate as well. Once the areas needing improvement have been identified, a plan can be developed to improve the work environment.

The management style of a given manager is much more difficult to address. However, if a certain manager has a high turnover rate, a higher complaint rate than other units, or the work is not getting done, it may be worth assessing this individual's management style. Even when negative documentation is brought to the attention of higher management, they usually have a difficult time dealing with it.

To illustrate some of the similarities between computer crime and espionage, there are two cases I wish to review. The computer crime case is Donald Gene Burleson, former employee of USPA & IRA Co., a Fort Worth securities trading firm, convicted in September 1988 of planting a virus that destroyed 168,000 sales commission records of his former employer. The espionage case is Edward Lee Howard, the former Central Intelligence Agency (CIA) employee, who flunked four polygraph tests prior to assignment to Moscow, and was fired by the agency. In 1984, he revealed the CIA operation in Moscow to the KGB. Both cases involve employees with personal problems which were known to the employers. When their employers fired them, the results were rather serious for both organizations.

Burleson was described as someone who denounced authority, believed federal income taxes were unconstitutional, and claimed he had not paid any since 1970. He complained that his salary was too low and had heated arguments with his superiors. A former co-worker stated, "he was so fanatical about everything...he could do anything with a computer" [4,64]. Burleson's virus, by destroying the commission records, held up the pay checks of employees. According to the assistant district attorney who prosecuted him, ..." 'Burleson was working on the virus every time he got mad--he was having conflicts with supervisors and people at work.'" Three days after the company fired him, he was able to enter the building at 3 a.m. and " 'manually activated his program with a second virus set to go off the next month--in case they found the first one, the other would go off later.'" While he tried to erase his tracks, he was not completely successful. He also mentioned his actions to a friend, another programmer [6].

Howard was a former Peace Corps volunteer and had worked for the Agency for International Development. After completing his initial CIA training, he was to be assigned to Moscow. The agency routinely polygraphs employees before such an assignment [8,75]. His polygraph, repeated four times, indicated he was being deceptive on two issues: theft and alcohol use. (He had previously admitted to a theft, but it was under the threshold used by the agency at that time. He admitted to repeated illegal drug use and there were indications that the agency was aware of his alcohol abuse.) Management had three options: fire him, assign him to a less sensitive Washington position where he could be monitored, or send him to Moscow. Management chose to ask for his resignation without explanation. Although the agency provided some "out placement" assistance, Howard took it hard as this was really the first failure of his life as he saw it at age 31 [8,81-87]. Within months after the firing, he initiated contact with the KGB, and two years later, ultimately fled to Moscow, while under FBI surveillance [8,224].

In both cases we have employees with personal problems which are known to their employers, given considerable access to sensitive systems or information, who ultimately turned against their organizations and in Howard's case against his country. In both cases, they should not have gotten as far as they did. Their organizations should have determined that they were not suitable for the positions of trust they occupied. When the securities firm determined that Burleson had to be separated, it could have made certain he had no access to the building or computer system. In the CIA case, the alternative of a Washington assignment seems to have been a more judicious decision. This alternative might have averted the disclosure which compromised the agency's operation in Moscow and resulted in the death of at least one agent, a Soviet citizen [8,249].

In addition to revenge against their former organizations, what rewards and penalties did they get? Burleson received no financial gain. He was fined $12,000 as a result of a civil suit brought by the former employer. He could be sentenced to up to 10 years in prison for the criminal actions [6].

Howard is believed to have received $150,000 from the KGB which was deposited in a Swiss bank account [8,221]. His penalty so far has been self-imposed exile from the U. S. If he ever returns to this country, is tried, and convicted, he would probably receive a life sentence [8,231].

In summary, what are the similarities between computer crime and espionage? They both require people and they both have negative impacts on the organizations effected. They both occur because someone subverted the controls that were intended to protect something of value.

Are we locked into the acronym MICE (money, ideology, compromise, and ego) to explain the motivation for espionage? No, relationships and ego may be the most important issues. For computer crime, is it always the money? Money is often a strong motivation, but there may be other reasons resulting from personal problems or unhappiness at work. We need to make certain that our EAP programs are working properly. We also need to review our organizations' internal environmental factors and the management styles or our managers. At the same time, we must make sure the controls are working.

References

[1] Allen, Thomas B ., and Polmar, Norman. Merchants of Treason: Americas Secrets for Sale. New York: Delacorte Press, 1988.

[2] Baker, Lara. "Threats to DOE Computers: A Perspective" in Summary of Papers to be Presented at the U. S. Department of Energy Computer Security Group Conference. GPO. 1988.

[3] Bologna, Jack. "Computer Crime: The Who, Where, When, Why, and How: Part 2, Data Processing and Communications Security, Spring 1986, pgs. 25-30.

[4] "Is your Computer Secure". Business Week, August 1, 1988, Pgs. 64-70.

[5] Moore, Lonnie. "Espionage and Computers". Presentation at U. S. Department of Energy Eleventh Computer Security Group Conference, Kansas City, MO, May 2-5, 1988.

[6] "Virus Conviction Makes News" DATAPRO Reports on Information Security Vol.4, No. 11, November 1988. Pgs. IS-99-801-111 thru 112.

[7] U. S. Department of Health and Human Services, Office of the Inspector General. Computer Related Fraud in Government Agencies - Perpetrator Interviews. Washington: GPO, 1985.

[8] Wise, David. The Spy Who Got Away. New York: Random House, 1988.

* The views expressed are those of the author and do not necessarily represent the Department of Veterans Affairs. *

BIOGRAPHY

Lloyd F. Reese is Chief of the ADP Security Division within the Office of Information Systems and Telecommunications at the Veterans Administration. The division is responsible for developing an effective ADP Security program within OIS&T and for developing policy and guidelines for a VA-wide program. Mr. Reese is a member of the Information Systems Security Association and currently the Vice-President for the National Capital Area Chapter. He is also a member of the American Society for Industrial Security. Since 1985 he has served on the ASIS Wa hington, D.C. Chapter's Subcommittee on Terrorist Activities which has presented a seminar on counter-terrorism each year. He teaches computer security courses for the USDA Graduate School. He holds a master degree in both justice and public administration from The American University.

A Summary of Computer Misuse Techniques

I eter G. Neumann and Donn B. Parker[1]
SRI International
Menlo Park CA 94025-3493

12th National Computer Security Conference
Baltimore, Maryland 10-13 October 1989

Abstract

We consider here general classes of computer misuse, including intentional security abuses and accidental misuses. The classification approach is intended to provide a basis for methodological threat analysis that assesses the significance of vulnerabilities in specific systems and networks. It is intended to increase the understanding of exploitable abuse techniques, and thereby to aid in reducing both the number of vulnerabilities and their seriousness.

Introduction

Security of computer systems and networks has developed without sufficient attention to actual loss experience. This becomes apparent in examining the literature on security policy and safeguards, where little is stated about specific abuses that must be defended against. Authors of security literature usually have not investigated loss experience, much less interviewed abusers. Experience indicates that computer misusers do not attack where controls and system security policy are strongest, but rather where vulnerabilities exist. Experience also suggests that varying certain characteristics of the user environment can increase the work and danger for the misusers.

We classify techniques involved in computer system misuse based on about 3,000 cases collected since 1970. The main purpose of the study is to provide a detailed resource for people involved in computer security, such as trusted system developers, testers, evaluators, certifiers, and researchers, including those working with formal models, specification, and verification. We hope that greater awareness and understanding of the techniques described herein will lead to systems that can be used more securely, with fewer opportunities for misuse.

Analysis suggests that the three commonly cited abuse categories (improper disclosure, modification, and denial of service, often related to losses of confidentiality, integrity, and availability, respectively) are greatly oversimplified. (Destruction is also often cited, although it is a combination of improper modification and denial of service.) Many abuses actually involve combinations of these categories, such as Trojan-horse and playback attacks. Others transcend these categories, for example, misrepresentation, impersonation, inferences that permit the derivation of data not even represented internally, and failure to act appropriately. In addition, misuse of conferred authority is often not addressed. In order to overcome this deceptive simplification, we consider nine classes of abuse and various types of abuse techniques that illustrate those classes.

This paper assumes a basic familiarity with computer security. Our terminology is generally consistent with the National Computer Security Center glossary (Glossary [88]), and we have chosen to avoid a proliferation of definitions by referring readers to that document.

[1] Copyright 1989 Peter G. Neumann and Donn B. Parker

Sources of Computer System Misuse

As noted in Neumann [88], there are three basic gaps that computer misuses can exploit:

- (1) *The technological gap* between what a computer system is actually *capable of enforcing* and what it is *expected to enforce* (e.g., its policies for data confidentiality, data integrity, system integrity, and availability). This gap includes deficiencies in both hardware and software (for systems and communications) as well as in their administration, configuration, and operation. For example, discretionary access controls such as user/group/world are intended to limit access, but are incapable of enforcing copy protection. Furthermore, flawed operating systems may permit violations of the intended policy.
- (2) *The sociotechnical gap* between *computer policies* and *social policies* such as computer-related crime laws, privacy laws, and codes of ethics. This gap arises when the socially expected norms are not consistent with computer policies. For example, issues of *intent* are not addressed by computer security policies, but are relevant to social policies.
- (3) *The social gap* between *social policies* and *actual human behavior.* This gap arises when people do not act according to expectations. For examp!e, authorized users may easily diverge from the desired social policies.

The technological gap can be narrowed by properly administered computer systems and networks that are meaningfully secure (e.g., that in part observe the criteria of the Orange Book and Red Book -- TCSEC [85] and TCSEC-TNI [87], respectively -- and the many U.S. National Institute of Standards and Technology federal information processing standards on security), at least to the extent of protecting against known vulnerabilities. The sociotechnical gap can be narrowed by well-defined and socially enforceable social policies, although computer enforcement still depends on narrowing the technological gap. The social gap can be narrowed to some extent by narrowing the first two gaps, with some additional help from educational processes. Malicious misuse of computer systems can never be prevented completely, particularly when perpetrated by authorized users. Ultimately the burden must rest on better computer systems and networks as well as better management and, to the extent possible, on the self-imposed discipline of information managers and computer users.

The primary focus here is on violations that exploit the technological gap. Approaches to avoidance, prevention, deterrence, detection, and recovery (whether in real time or after the fact) are fundamental to closing that gap, and these are also discussed. Reducing the other two gaps is also important, though treated here only superficially.

Classes of Techniques for Computer System Misuse

Computer misuse techniques are here classified according to Figure 1. For visual simplicity, the figure is approximated as a simple tree. However, it actually represents a system of descriptors rather than a taxonomy in the usual sense, in that a given misuse may involve multiple techniques within several classes.

The order of categorization depicted is roughly from the physical world to the hardware to the software, and from unauthorized use to misuse of authority. The first class includes external misuses that can take place without any access to the computer system. The second class concerns hardware misuse, and generally requires some involvement with the computer system itself: two examples in this class are eavesdropping and interference (usually electronic or electromagnetic, but optical and other forms are also possible). The third class includes masquerading in a variety of forms. The fourth includes the establishment of deferred misuse, for example, the creation and enabling of a Trojan horse (as opposed to subsequent misuse that accompanies the actual execution of the Trojan-horse program -- which may show up in other classes at a later time). The fifth class involves bypass of authorization, possibly enabling a user to appear to be authorized -- or not to appear at all (e.g., invisible to the audit trails). The remaining classes involve active and passive misuse of resources, inaction that might result in misuse, and finally misuse that helps in carrying out additional misuses

```
                     /\
                    /  \
1. External        /    \ Computer system access
   misuse                /\
                        /  \
   2. Hardware         /    \ Computer system use
      misuse                 /\
                            /  \ Apparently authorized use
   3. Masquerading         /    \ (even if clandestine)
                                 /\
                                /  \
   4. Setting up sub-          /    \ Direct use
      sequent misuse                 /\
                                    /  \ Use apparently conforming
   5. Bypassing intended           /    \ with intended controls
      controls                           /\
                                        /  \
          6. Active misuse             /    \ Active use
             of resources                    /\
                                            /  \
           7. Passive misuse               /    \ Apparently normal use
              of resources                       /\
                                                /  \
             8. Misuse resulting               /    \ Apparently proper use
                from inaction                        /\
                                                    /  \
                9. Use as an aid                   /    \ Proper use
                   to other misuses
```

Figure 1: Classes of Computer Misuse Techniques

(such as preparation for an attack on another system, or use of a computer in a criminal enterprise).

The main downward sloping right-hand diagonal line in Figure 1 indicates typical steps and modes of intended use of computer systems. The leftward branches all involve misuse, while the rightward branches represent potentially acceptable use -- until a leftward branch is taken. (Each labeled mode of usage along the main diagonal intended-usage line is generally the antithesis of the corresponding leftward misuse branch.) Every leftward branch represents a class of vulnerabilities that must be defended against, that is, detected, avoided, and/or recovered from. The means for prevention, deterrence, avoidance, detection, and recovery typically differ from one branch to the next.

To reiterate, the tree in Figure 1 relates to the classification of technique types. Actual misuse often involves multiple misuse types, with one misuse enabling another. For example, the West German Chaos Computer Club members who attacked NASA systems on the SPAN network used (at least) techniques of external misuse, masquerading, Trojan-horse attacks used to capture passwords, bypass of intended controls, failure of system administrators to act prudently, and both active and passive misuse of resources. (References to this case and to most of the other cases mentioned here are given in Neumann [89].)

Types of Computer Misuse

Representative misuse techniques are sketched below for each class. While the basic classification system is thought to be fairly comprehensive, new techniques and subcases are likely to be discovered as technology advances. On the other hand, most of the attack methods being used today are merely variants of techniques that have been known for years. Indeed, actual loss experience shows that system and network problems that facilitate attacks are reincarnated in new systems and networks, although the details may change somewhat.

1. *External misuse* -- Generally nontechnological and unobserved, physically separate from computer and communication facilities: physical scavenging (e.g., collection of waste paper or

other externally accessible computer media such as discards), visual spying (e.g., remote observation of typed keystrokes or screen images), and deception (e.g., misrepresentation) external to the computer systems and telecommunications. These techniques have no directly observable effects on the systems and are usually undetectable through computer security systems; however, they may lead to subsequent technological attacks, and thus are vital to the identification of security vulnerabilities.

2. *Hardware misuse* --

   - (a) Passive hardware misuse, with no (immediate) side effects on hardware or software behavior: electronic or other eavesdropping and logical scavenging. Eavesdropping may be carried out remotely (e.g., by picking up emanations) or locally (e.g., by planting a spy-tap device in a terminal, mainframe, or other hardware). Logical scavenging may involve examination of discarded computer media.

   - (b) Active hardware misuse, with side effects: theft of computing equipment and physical storage media; physical attacks on equipment and media; hardware modifications such as internally planted Trojan-horse hardware devices; interruption of or tampering with power supplies or cooling; and interference (electromagnetic, optical, or other). These activities have direct effects on the computer systems (e.g., internal state changes or denials of service)

3. *Masquerading* -- Impersonation; playback and spoofing attacks; piggybacking on other users; and telephone-network weaving to hide dial-up origin (as in Stoll [87]). These activities may be indistinguishable from legitimate activity.

4. *Setting up subsequent misuses* -- Planting and arming software Trojan horses with techniques such as logic bombs and time bombs, letter bombs, malicious worms, and viruses. The setting up of these so-called "pest" programs may actually employ misuses of other classes such as bypasses or misuse of authority, or may be planted via completely normal use, as in a letter bomb. The subsequent execution of the deferred misuses may also rely on further misuse techniques. Alternatively, it may simply involve the occurrence of some logical event (e.g., a particular date and time, or a logical condition), or rely on the curiosity, naïveté, or normal behavior of the victim. Indeed, because a Trojan horse typically executes with the privileges of its victim(s), its execution may require no further privileges. For example, a Trojan horse program might find itself authorized to delete all the victim's files. A Trojan horse letter bomb (with hidden control characters and escape sequences squirreled away in the text) might be harmless unless explicitly read interpretively or otherwise executed; however, if the system permits the transit of such characters, the letter bomb would be able to exploit that flaw and be executed unbeknownst to the victim.

5. *Bypass of intended controls* -- Circumvention of existing controls or improper acquisition of otherwise denied authority, presumably with the intent to subsequently misuse the acquired access rights. Common cases of unauthorized access result from system and usage flaws (e.g., trapdoors that permit devious access paths) such as improper domain initialization, improper encapsulation, inadequate information hiding, incomplete deallocation (e.g., storage or access control residues), incomplete interrupt or error handling, naming problems such as search-path anomalies and inconsistent aliases, and lack of adequate validation. Tailgating may occur accidentally when a user is randomly attached to an improperly deactivated resource such as a port through which a process is still logged in with its original user no longer attached. Unintended access may also result from other trapdoor attacks, logical scavenging (e.g., reading a scratch tape before writing upon it), and asynchronous attacks (e.g., incomplete atomic transactions, and discrepancies between time of check and time of use). For example, trapdoors in the implementation of encryption can permit unanticipated access to unencrypted information. *Password attacks* are a particularly insidious subclass and may involve, for example, guessing of common passwords (dictionary words, initials, proper names); capture of unencrypted passwords in transit (via local or global net, or by UNIX /dev/mem), whether or not they are stored in encrypted form; derivation of passwords (exhaustively, algorithmically, by inference, by pre-

encryptive dictionary attacks as in Morris and Thompson [79], quitting during login with a wrong password and discovering oneself logged in; discovery of unintentional universal passwords (e.g., Young and McHugh [87]); editing an inadequately protected password file to insert a bogus user identifier and password; and inserting a trapdoor into the login program by Trojan horsing the compiler (Thompson [84]). The variations within this class are amazingly rich.

6. *Active misuse of resources* -- Misuse of (apparently) conferred authority that alters the system or its data. Examples include misuse of administrative privileges or superuser privileges; changing access controls to enable other misuses of authority; harmful data alteration and false data entry; denials of service (including saturation, delay, or prolongation of service); and the somewhat exotic salami attacks in which round-off is collected (for personal or corporate gain). Note that in Classes 6 and 7 the apparently conferred authority may have been obtained surreptitiously, but otherwise appears as legitimate use.

7. *Passive misuse of resources* -- Misuse of (apparently) conferred reading authority, such as browsing (without specific targets), searching (for specific patterns), access to data aggregates that are more sensitive than the individual items, drawing inferences (e.g., as in traffic analysis), and exploitation of covert channels (storage or timing channels). These events have no appreciable effect on the objects used or on the state of the system (except of course for the execution of computer instructions and the resulting audit data). They need not involve unauthorized use of services and storage. Note that certain events that superficially might appear to be passive misuse may in fact result in active misuse -- for example, through time-dependent side effects.

8. *Misuse resulting from inaction* -- Failure to avert a potential problem in a timely fashion, or an error of omission, for example. This class might be considered as a limiting case of passive misuse; however, it seems qualitatively different and thus is distinguished as a separate class. An accidental example arose in the Air Force's public resale of magnetic tapes without their first having erased the contents. Intentional misuse would result from someone detecting but not reporting a serious security flaw, e.g., saving that knowledge for a possible subsequent abuse.

9. *Use as an indirect aid in committing other misuse* --

   - (a) As a tool in planning, developing, controlling, or carrying out computer-system misuse, such as seeking matches in the encrypted password file by preencrypting dictionaries and likely passwords (the eventual attack is noted in Class 5 above); searching with an autodialer for answering modems; seeking to determine flaws in a system design and implementation for future exploitation by conducting black-box ("Gedanken") experiments without any internal knowledge; factoring very large integers to break public-key encryption schemes; or analyzing database query responses for inferences. Activities of this subclass may subsequently lead to computer misuses of other classes. Note that each of these activities could be aimed at attacking a computer system other than the one on which the indirect misuse is carried out. Each of these activities may seem suspicious, but is not necessarily yet an overt abuse. A particularly subtle example of this class might be called *anticipatory anomaly detection training*, by which a user slowly alters his "normal" legitimate behavior patterns in the hope that an adaptive anomaly detection system will train itself to accept behavior and so miss an actual attack. (Class 4 bears some resemblance to Class 9(a); however, the Class 4 activities may have a direct effect on the target system, while the Class 9(a) activities may not yet imply a compromise.)

   - (b) As a tool in planning, developing, controlling, or engaging in criminal enterprise (e.g., managing an illegal drug business, or committing financial fraud), or performing unethical acts (e.g., misuse of company resources for private purposes).

Many of the intentional computer abuses have accidental counterparts. For example, eavesdropping, interference, piggybacking, tailgating, false data entry, and inaction all may occur accidentally without specific malicious intent; the discovery of their feasibility might then inspire subsequent intentional abuse. Thus, we make an informal distinction here between "abuse" and "misuse", using *abuse* to refer to intentional acts, and *misuse* to refer more generally to accidental or intentional acts. The classification addresses both intentional abuses of computers and corresponding accidental misuses, primarily from the vantage point of security; however, we note that there are other accidental forms of misuse that are not represented here -- for example, some that compromise human safety or the functional correctness of the application.

An informal distinction is also made between unauthorized use (Classes 1 through 5 above, more or less) and misuse of conferred authority (Classes 6 through 9). Note, however, that a masquerader or penetrator may become essentially indistinguishable from a legitimate user, having gained what appears to be authorized access. Furthermore, the activities of Class 4 may be either unauthorized or within authority, but nevertheless malevolent. In many cases it may not be particularly helpful to try to distinguish between a penetrator and a legitimate user, particularly when either user could be misusing authority -- however it was conferred.

Anderson [80] has previously characterized categories of threats, roughly comparable to the present classification as follows: external abuse (Classes 1 and 2), masquerading (Class 3), clandestine activities (Classes 4 and 5), and misfeasance (performing an authorized action in an improper way -- Classes 6 through 9). We have thus seemingly subdivided three of his categories and provided specific types within classes. However, there seems to be considerable intrinsic ambiguity even in the most carefully constructed definitions. For example, authorization is not always clearcut; glaring system flaws may beg the question of what is proper use, particularly for those flaws identified as "features". Once having penetrated a system, a masquerader appears as if authorized. Furthermore, as noted above, the Class 4 techniques typically may involve clandestine activity and misfeasance, and of course may also employ techniques of other classes in the execution of further abuses. Thus we expect that the classification approach given here will not be the last word. (An earlier discussion of various types of system vulnerabilities within what corresponds roughly to Class 5 is found in Neumann [78], inspired by earlier work at the USC Information Sciences Institute by Bisbee, Carlstedt, Hollingworth, et al., who sought to build tools that searched for specific types of flaws. Classes of abuse are also considered in Denning and Neumann [85], with respect to anomaly detection.)

This study draws on extensive experience with computer abuse over twenty years. One of the authors has been collecting computer abuse cases at SRI since 1970, and has been involved in the identification, study, and reporting of abusive techniques (e.g., Parker [72], [76], [83]) including work for the criminal justice community (Parker [89]). The other author has collected many cases of computer misuse, including those in which security, reliability, human safety, or financial well-being were seriously at stake. (See Neumann [89] and the on-line ACM Forum on Risks to the Public in the Use of Computers and Related Systems.)

Collaborative Misuse

Most of the abuses noted above can result from the actions of just one person. Others may require some collusion. In general, intentional collusion can arise with different individuals and different techniques. Note that successful Trojan horses may require the unwitting collaboration of the victims, but abuse by only one user.

Through the use of various compartmentation techniques and multiperson authorizations, it is possible to hinder the abilities of single individuals to perform certain abuses. In addition, periodically changing the application of these controls makes the field of attack more unpredictable and somewhat more difficult for the attackers. For example, the principle of separation of duties (both statically and dynamically) requires different user roles for different purposes; the principle of least privilege requires allocating only the needed privileges for any given role (including withholding privileges altogether when appropriate). Suppose, however, that separation of duties is practiced carefully throughout (e.g., in the design, implementation, configuration, maintenance, operation, and use of the systems and networks). As a consequence, certain abuses would then require collaboration to succeed. Indeed, as we progress to better computer systems and better administrative practices that enforce separation of duties, the necessity for -- and the likelihood of -- misusers resorting to collaborative abuses can be expected to increase accordingly.

Effects of Computer Misuse

Misuse may include various forms of unauthorized reading, writing, copying, deleting, and executing -- including logical theft of computational resources. It may be detectable or undetectable. It may result in deaths and injuries, compromises to national security and global survival, loss of personal privacy or constitutional rights, financial fraud, or destruction of property, to name just a few critical areas. Misuse may also involve loss of real-time control, rigging of computer-controlled elections, loss of safety in medical applications, or loss of security in scientific or business computing and communications -- including electronic mail. Attempts to enumerate all of the possible effects would be futile, although many examples are included in Neumann [89], Parker [72], Parker [76], and Parker [83].

From the victim's perspective, consequential losses usually result from direct losses caused by misuse. In many cases the consequential losses exceed the direct loss, although the attacker may have intended to inflict direct losses rather than consequential losses. Consequential losses include the costs associated with recovery of resources and system availability; correction or replacement of security controls and removal of security flaws; insurance claim efforts and increased cost of insurance; loss of credibility and public image (e.g., reflected as a loss of credit rating or customers); special audits; litigation; removal and replacement of perpetrators; and staff time spent in discussions and wheel-spinning.

Motivations for Computer Misuse

While there are many purposes and motives behind computer-related abuses, a detailed sociological or psychological classification is beyond the scope of this study. Nevertheless, it is useful to illustrate the diversity of motives. Typical intentional purposes include espionage (corporate, national, and international), financial gain, fraud, theft, piracy, violation of contractual agreements, intellectual challenge, revenge, threats, blackmail, and extortion. Typical causes of unintentional misuse include curiosity, boredom, laziness, ignorance, misguided intent, incompetence, and inattentiveness, among others.

Contrary to the popular belief that computer crime is motivated by greed and high living, criminological studies and SRI interviews of over 100 computer criminals (Cressey [71], Parker [76], and Parker [89]) suggest that primary motivations include the following, sometimes in combination: (1) the need to resolve intense personal problems such as job-related difficulties, mental instability, debt, drug addiction, loneliness, jealousy, and desire for revenge; (2) peer pressures and other challenges, for example, among malevolent hackers; (3) idealism or extreme advocacy, for example, by espionage agents and terrorists; and (4) financial gain. Cases 1 and 2 apply largely to amateur white-collar criminals and misguided individuals, while case 4 is more applicable to career criminals and insiders; case 3 seems to represent a mixture of people. Computer abuse *per se* is often a secondary consideration. However, the opportunities for personal financial gain are considerable today, particularly among authorized users and a few masqueraders, and thus the need for better security controls and administration is very pressing.

Skills and Knowledge Required

Each of the previously noted techniques requires an associated range of skills and knowledge for its execution. For example, the discovery of a trapdoor may require considerable sophistication, while its exploitation may be relatively easy. The technical skill levels required for some types of abuse are summarized below in general terms (see Parker and Dewey [78]). Skills may also include programming ability, hardware knowledge, communications expertise, and interpersonal suavity. Some technical knowledge of the target systems is frequently required. The level of skills and knowledge required may be approximately associated with the abuse techniques, as follows:

- *Few, if any, technical skills or knowledge required*: misrepresentation; visual spying; physical scavenging and thieving; physical attacks on equipment; random interference; false data entry; external collusion.

- *Some technical skills and knowledge required in some cases*: browsing and searching; logical scavenging; inferring, aggregating, traffic or activity monitoring; selective interference; eavesdropping; leaking data; impersonating; playback attacks; piggybacking; misusing authority; improper reading, writing or copying; integrity violations; denying use; letter bombing; trap door exploitation; network weaving; internal collusion.
- *Greater technical skills and knowledge generally required (at least in new attacks, but not so much in copycat cases)*: system alterations; exploitations such as Trojan horses, logic bombs, time bombs, worms and viruses; incremental attacks; asynchronous attacks; hardware modifications.

In general, it is dangerous to assume that the requisite skills and knowledge are not available. In particular, former employees and disgruntled or dishonest current employees usually have abundant skills and knowledge. This again illustrates the importance of varying certain control parameters so that although abusers may know the existence of the controls they may not know the current settings. In menu-driven and self-prompting systems, however, any lack of knowledge can often be quickly overcome. Deterministic digital technology can be analyzed, even without documentation.

Resources Required

The resources required for computer misuse vary widely, depending on the techniques used and the skills of the perpetrator. Surprisingly few additional resources may be needed in some cases -- for example, for the disgruntled employee. In other cases, extensive resources may be employed -- as in the example of collaborative efforts to factor 100-digit numbers, which in one case required about 40 MIP-years of computing distributed across many different machines (in a few weeks!). As noted above, use of the target computer may not be necessary. Documentation (manuals, object code, source code) may help in some cases, and may be unnecessary in others. Possession of personal computers and modems is useful in some cases. In general, it is dangerous to assume that adequate resources are *not* available to would-be perpetrators. Equipment can be stolen; software can be down-loaded from pirate bulletin boards or acquired without authorization; telephone services can be obtained through toll fraud.

Avoidance, Prevention, Detection, and Recovery

Efforts should be made to narrow each of the three gaps discussed above. As noted above, our primary emphasis here is on the technological gap, which can be reduced dramatically by computer systems with better security and better system administration.

Each of the classes in Figure 1 has its own set of countermeasures for coping with misuse, and its own tradeoffs. For example, external abuse and passive hardware abuse may be very difficult to detect; thus, if they represent a sufficient threat, additional effort may be needed to prevent them. Where such threats are perceived, defensive methods may include proper disposal of discarded media and shielding to prevent emanations. Active hardware abuse such as intentional or accidental interference may be relatively easier to detect, but also requires considerable foresight to prevent. The remainder of the classes require computer system software and administrative countermeasures appropriate to the individual threats, although there is considerable commonality within each class -- as in defending against Trojan horses, viruses, and other pest programs.

Avoidance of Misuse

Avoidance is a security function that is often overlooked; because it is so obvious, security practitioners falsely assume that others have already considered it. Avoidance can be applied quite simply: e.g., remove the threats from assets subject to loss, to make the misuse techniques more difficult to use; remove the assets from the threats to make access impossible or impractical; redefine the security problem to give both responsibility and authority to parties better qualified or better motivated to deal with it.

Prevention or Deterrence of Misuse

With respect to computer operating systems and application software, enforcement of mandatory security (e.g., Bell and La Padula [76]) and some form of multilevel integrity (such as Biba [75]) can reduce the potential for misuse considerably, even in unclassified applications (see Lipner [82]). Observance of the criteria of the Orange Book (TCSEC [85]) and Red Book (TCSEC-TNI [87]) also can contribute significantly to security. The principle of separation of duties and the principle of least privilege are fundamental (e.g., see Clark and Wilson [87]), and can hinder both single-person misuse and collusion. In addition, multiperson authorizations may be desirable where collusion is expected to be a problem -- requiring not only separation of duties but also explicit mechanisms for joint authorization.

The consistent use of good software engineering practice coupled with well-conceived programming languages (e.g., modular systems, strong typing, use of abstraction and encapsulation of data types, separate compilation, run-time checking, and systematic exception handling) can contribute significantly to the security, reliability, and safety of applications as well as systems. In particular, many characteristic security flaws can be avoided altogether, or significantly minimized, particularly those in Class 5 (bypass of controls) as discussed in Neumann [78]. For example, the Internet worm atta ' exploited trapdoors in the *debug* option of *sendmail* and in the *gets* program called by *fingerd* in BSD-derived versions of UNIX[2] (see Spafford [89], Rochlis and Eichin [89], Seeley [89]). Both of these trapdoors could have been avoided by the judicious use of software engineering, particularly with some bounds-checking and application of the principle of least privilege.

One goal is to make the established security policy as close as possible to the actual intent as to what should be accessible, thereby narrowing the technological gap. Another goal is to make the misusers' targeted environments as unpredictable as possible -- without confusing normal users. Thus, some variability can provide both deterrent and preventive effects. Narrowing the sociotechnical gap requires better laws and codes of ethics, but ultimately the social gap suggests that the sociotechnical gap cannot be closed without more realistically enforceable security policies that permit the narrowing of the technological gap. Because some hostile users must be assumed to exist, laws and codes of ethics for computer use are of limited value. Ultimately, the burden in critical systems rests on narrowing the technological gap to combat both untrusted users and trusted abusers, and also on the use of audit-trail analyses seeking to identify both penetrators and authorized-but-untrustworthy users who cannot be controlled directly.

Detection and Identification of Misuse

The systematic analysis of well-supported audit trails appears to be a rapidly growing and very promising field of endeavor. (Lunt [88] provides a survey of various systems currently in use or under development.) Real-time identification of likely computer misuse -- including misuses of authority -- will be of enormous importance in the future. Although it is too early to assess the effectiveness of today's systems, both statistical and expert-system rule-based approaches are being explored. (Initial efforts in credit card applications have been relatively useful.)

Real-time anomaly detection is potentially applicable for many of the abuse classes, particularly Classes 2 through 7, as well as in some types within Class 8. Activities of Class 9 would be detectable only when performed on systems being monitored. (Detection of attempts to compromise the anomaly detection mechanisms themselves would of course be of particular interest!)

To the extent that accidental misuse may appear similar in nature to intentional abuse, detection of accidental misuse should also be of interest to audit-trail analysts. The anomaly detection approach is applicable to a wide variety of computer misuses, not just to violations of security (such as technological gap compromises of confidentiality, integrity, and availability), but also to illegal, unethical, or simply questionable activities, e.g., to monitoring second- and third-gap activities. In some cases it may also be used to detect or even to block accidental misuses. Interviews with perpetrators reveal that two great fears are unexpected detection of misuse

[2] BSD is an acronym for Berkeley Software Distribution; UNIX is a registered trademark of AT&T Bell Laboratories.

activities and loss of anonymity (see Parker [83], [89]). Detection capabilities are extremely important in both of these cases.

Recovery from the Effects of Misuse

Whether the results of misuse are successful, partially successful, or abortive (from the viewpoint of the perpetrator), recovery must be an integral part of the security process. In general, it should be the first function applied in order to minimize further loss occurring before other functions have been applied. Efforts have often been restricted to physical disaster recovery; however, the epidemic of computer Trojan horse and virus attacks has demonstrated the importance of recovery from misuse techniques that in the final analysis are not totally amenable to technological means of prevention or detection.

The generally accepted disaster recovery or business resumption planning efforts in systems operation must be extended to deal with logical disasters as well as physical disasters. One of the increasingly popular ways of doing this is to create a technological crisis team that can cope effectively with the misuse techniques discussed here. Clearly the gamut of misuse techniques must be considered.

Usefulness of This Classification Approach

The ultimate goal here is to achieve better security against all realistic threats that can be effectively addressed. Analysis of the abuse cases shows that both accidental and intentional perpetrators tend to cause losses where controls are absent or weak. It is generally less fruitful for the rational, intentional misuser to attack where defenses are strongest; therefore, security requires continually searching for and correcting significant vulnerabilities -- ideally, before they are discovered by the would-be attackers. In general, it is desirable to apply well-known controls to protect from well-known threats, according to a standard of due care; the remaining vulnerabilities should be addressed according to the greatest potential exposure to perpetrations, in terms of would-be perpetrators with the necessary skills, knowledge, access, resources, and motives. It is important to remember that defensive measures must withstand misuse by rational, willful attackers, irrational perpetrators, and accidental misuse; must meet a standard of due care; and be cost-effective with respect to the threats they address.

It is generally of limited value to attempt to quantify risks in terms of probabilities of attack, because of the extent and complexity of the combinations of misuse techniques, the lack of independence among different variables, the impossibility of predicting perverse human behavior on a small-scale basis in limited populations, and other uncertainties such as consequential losses. Nevertheless, it is worth noting that among the collected cases of misuse considered, the most prevalent classes were (in order of decreasing frequency) active misuse of authority (by far the most common), masquerading, bypassing of intended controls, setting up subsequent misuses, hardware misuse, passive misuse of authority, and external misuse. The remaining two classes (8 and 9) were much less evident (although their presence is at the same time harder to detect). Note that many cases involved multiple classes of techniques.

We hope that our effort will be a significant aid in the identification of material and intangible vulnerabilities, and will thereby help to increase security coverage by providing a comprehensive methodological approach to measuring the effectiveness of the policies, systems, security controls, and practices with respect to abuse techniques.

Conclusions

Because abuses may exploit various combinations of techniques, it is important to visualize the set of techniques discussed here in the context of complete abusive events. We have considered here most types of computer system and network misuses that have been exploited. Most of them can also be expected in the future. As computer technology becomes more widely demystified, the knowledge of how to perform attacks will become increasingly widespread. It is thus vital that considerable effort be expended to narrow each of the three gaps noted above as sources of vulnerabilities. However, the intrinsic limitations on technology and predicting human behavior must be taken into account, along with the social implications of computer security (e.g., Denning et al. [87]).

There is a significant danger in not being aware of -- and not eliminating or narrowing -- vulnerabilities that are known only to selected subcultures within the computing community. For example, various computer and communications vendors were not seriously victimized internally by the Internet worm -- partially because each had recognized the vulnerabilities and had developed code modifications or administrative practices to limit the consequences. Unfortunately, this constructive knowledge was not propagated to their customers. A variety of factors could have been involved -- e.g., vendors may have been reluctant to publicize or emphasize the vulnerabilities for fear of attack and, even if they had distributed fixes, their customers might have had little motivation to install those changes (particularly when only object code was available) unless alerted to the specific dangers -- which would have alerted would-be attackers as well. This type of dichotomy will continue to exist.

There is a long-standing argument about the extent to which knowledge about abuse techniques should be made available. On the one hand, there are many system vulnerabilities that can be exploited; thus, there is a risk that dissemination of such details could stimulate potential perpetrators to engage in harmful acts. On the other hand, experience shows that ignorance of these techniques by potential victims is even more harmful, because clever perpetrators generally can gain the knowledge they need -- whereas security administrators and systems people often cannot do so as easily, or are not trained to anticipate the diverse techniques and characteristics of perpetrators. Publication of knowledge about vulnerabilities and attack methods is likely to have a bene[illegible]al net effect by telling security specialists and potential victims what to expect, provided that it is accompanied by readily implementable countermeasures. Overall, better understanding of the vulnerabilities, better computer systems and networks, and better use and administration must go hand in hand.

References

Anderson [80], J.P. Anderson, ''Computer Security Threat Monitoring and Surveillance'', James P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

Bell and La P[illegible] [76], D.E. Bell and L.J. La Padula, ''Secure Computer System: Unified Exposition and Multics Interpre[illegible]'', ESD-TR-75-306, MITRE Corp., Bedford, Massachusetts, March 1976.

Biba [75], K.J. [illegible] ''Integrity Considerations for Secure Computer Systems'', Report MTR 3153, MITRE Corp., Bedford, Massachusetts., June 1975.

Clark and Wilson [87] D. Clark and D. Wilson, ''A Comparison of Commercial and Military Computer Security Policies'', *Pr[illegible] 1987 IEEE Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 184-194.

Cressey [71], D.R. C[illegible]ssey, ''Other People's Money -- A Study in the Social Psychology of Embezzlement'', Wadsworth Publishing, Belmont CA, 1971.

Denning and Neumann [85], D.E. Denning and P.G. Neumann, ''Requirements and Model for IDES -- a Real-Time Intrusion-Detection Expert System'', August 1985, Computer Science Laboratory, SRI International, Menlo Park, California.

Denning et al. [87], D.E. Denning, P.G. Neumann and D.B. Parker, ''Social Aspects of Computer Security'', *10th National Computer Security Conference*, NBS, Gaithersburg, Maryland, September 1987.

Glossary [88], National Computer Security Center, ''Glossary of Computer Security Terms'', NCSC-TG-004, Version-1, 21 O[illegible]ober 1988.

Lipner [82], S.B. Lipner, ''Non-Discretionary Controls for Commercial Applications'', *Proc. 1982 IEEE Symposium on Security and Privacy*, 26-28 April 1982, pp. 2-10.

Lunt [88], T. Lunt, ''Automated Audit Trail Analysis and Intrusion Detection: A Survey'', *11th National*

Computer Security Conference, Baltimore, Maryland, 1988.

Neumann [78], P.G. Neumann, ''Computer Security Evaluation'', *AFIPS Conference Proceedings* (National Computer Conference), AFIPS Press, January 1978, pp. 1087-1095. (Reprinted in Rein Turn, ed., *Advances in Computer Security*, Volume 1, Artech House, 1981.) (The article contains references to earlier work by Bisbee, Carlstedt, Hollingworth, et ai., at ISI.)

Neumann [88], P.G. Neumann, ''The Computer-Related Risk of the Year: Computer Abuse'', 3rd Annual Conference on Computer Assurance (COMPASS '88), National Bureau of Standards, 28-30 June 1988, pp. 8-12 (IEEE 88CH2628-6).

Neumann [89], P.G. Neumann, ''RISKS: Cumulative Index of Software Engineering Notes -- Illustrative Risks to the Public in the Use of Computer Systems and Related Technology'', *ACM Software Engineering Notes* 14 1, January 1989, pp. 22-26. (This index includes highlights from the on-line ACM Forum on Risks to the Public in the Use of Computers and Related Systems.)

Morris and Thompson [79], R. Morris and K. Thompson, ''UNIX Password Security: A Case History'', *Comm. ACM* 22 11, November 1979, pp. 594-597.

Parker [72], D.B. Parker, ''Computer Abuse'', Stanford Research Institute report, 1972.

Parker [76], D.B. Parker, ''Crime by Computer'', Charles Scribner's Sons, New York, 1976.

Parker [83], D.B. Parker, ''Fighting Computer Crime'', Charles Scribner's Sons, New York, 1983.

Parker [89], D.B. Parker, ''Criminal Justice Resource Manual on Computer Crime'', National Institute of Justice, U.S. Government Printing Office, Washington DC, 1989.

Parker and Dewey [78], D.B. Parker and R. Dewey, ''A Guide to EDP and EFT Security Based on Occupations'', Federal Deposit Insurance Corp. Division of Management Systems, Washington DC, 1978.

Rochlis and Eichin [89], J.A. Rochlis and M.W. Eichin, ''With Microscope and Tweezers: The Worm from MIT's Perspective'', *Comm. ACM* 32 6, June 1989, pp. 689-698.

Seeley [89], Donn Seeley, ''Password Cracking: A Game of Wits'', *Comm. ACM* 32 6, June 1989, pp. 700-703.

Spafford [89], Eugene H. Spafford, ''The Internet Worm: Crisis and Aftermath'', *Comm. ACM* 32 6, June 1989, pp. 678-687.

Stoll [88], Clifford Stoll, ''Stalking the Wily Hacker'', *Comm. ACM* 31 5, May 1988, pp. 484-497.

TCSEC [85], ''Department of Defense Trusted Computer System Evaluation Criteria'', DOD 5200.28-STD, December 1985 (''Orange Book'', ''TCSEC'').

TCSEC-TNI [87], ''Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria'', NCSC-TG-005 Version-1, 31 July 1987 (''Red Book'', ''TNI'').

Thompson [84], K. Thompson, ''Reflections on Trusting Trust'' (1983 Turing Award Lecture), *Communications of the ACM* 27 8, August 1984, pp. 761-763.

Young and McHugh [87], W.D. Young and J. McHugh, ''Coding for a Believable Specification to Implementation Mapping'', *Proc. 1987 Symposium on Security and Privacy*, IEEE Computer Society, Oakland, California, April 1987, pp. 140-148.

TRACK C

INTEGRATION OF SECURITY INTO THE ACQUISITION LIFE CYCLE

William Norvell, Ph.D.
Hughes Aircraft Company
P.O. Box 92919
Los Angeles, CA 90009

Abstract

The insecurity of many deployed systems argues the need to examine and improve the two Department of Defense (DoD) processes by which secure systems are defined, developed, and deployed. One process - accreditation - is driven by regulations and becomes, in practice but not by intent, the process of meeting regulations; the other process - acquisition - scarcely involves attention to security considerations. In general, these processes are conducted in parallel and independently of each other. The net result is that while some security results from the accomplishment of the accreditation and acquisition processes, many security requirements are not met.

The solution is to integrate accreditation activities into the acquisition process and to ensure that all security requirements are specified in the functional baseline for design and test. This solution forces security requirements, like any other set of system requirements, to be collectively treated "top-down" and addressed in each phase of the acquisition life cycle - concept exploration through operations support.

Deployed Systems Are Not Always Secure

Recent history offers several tragic examples of systems that did not meet security requirements.

Security Compromise. On May 19, 1979, John A. Walker was arrested attemping to pass 129 classified documents to Aleksey Tkachenko, a Soviet embassy official. For 20 years his gang had delivered to the KGB the Navy's sensitive submarine secrets. [1] Today, the Soviet Akula (Russian for shark) is the best submarine in the world and is grudgingly referred to as the "Walker-class" submarine. [2]

Security Integrity. On July 3, 1988, the Aegis system on the U.S.S. Vincennes could not distinguish the difference between a 62 foot F-14 Tomcat and a 177 foot Airbus on a regularly scheduled flight from Bandar Abbas to Dubai. The Iran Air Airbus was on course in a prescribed 20-mile wide air corridor, 27 minutes late, and the pilot's last words on civilian radio frequency were, "I am at one-two-zero [12,000 feet], climbing to one-four-zero [14,000 feet]." [3]

Security Denial of Service. On May 4, 1982, during the Falkland Islands conflict, a software bug in the frigate Sheffield's air defense system jammed the radar and could not pick up an incoming Exocet missile when the Sheffield's captain, Sam Salt, was on a communications hookup to naval headquarters. The unfortunately timed call allowed the craft to take a direct hit. [4]

The DoD processes that define, develop, and deploy systems to meet security requirements must be improved.

Accreditation Is a Regulation-Meeting Process

The first process, accreditation, is defined in DoD Directive 5200.28, Security Requirements for Automated Information Systems (AISs) [5], and is implemented in three key military regulations:

1) AFR 205-16, Air Force Automatic Data Processing (ADP) Security Policy, Procedures, and Responsibilities [6]
2) AR 380-380, Army Automated Systems Security [7]
3) OPNAVINST 5239.1A, Navy ADP Security Program [8]

Each regulation specifies accreditation activities, illustrated in Table 1, that culminate in obtaining the approval of a Designated Approving Authority (DAA) to process sensitive information in that authority's operational environment.

Table 1. OPNAVINST 5239.1A Accreditation Support Documentation

ADP Security Officer and System Security Officer Information
ADP Equipment Identification and Location
Interconnection Line Diagrams
Data Percentages versus Level and Type
Operating System Description
Application Software Description
Security Mode of Operation
ADP Security Operating Procedures
Risk Assessment
Countermeasure Descriptions

Previous System and Network Accreditations
Security Directives Compliance
Security Test and Evaluation (ST&E) Test Plans
ST&E Test Reports
TEMPEST Accreditation
Physical Accreditation
Contingency Plan
Contingency Plan Test Results
Activity ADP Security Plan
Other Documentation as Required by the ADP Security Officer

Government directives require that these activities be "considered throughout the life cycle of an AIS from the beginning of concept development until replacement." [5] But, in practice, accreditation is normally performed by the Government in an environment independent of system development. It becomes, even if not by the intent of Directive 5200.28 and military regulations, a document-producing process dedicated to convincing the DAA that the system should be accredited. That is regrettable, because the results of most accreditation activities could contribute directly to the definition, development, and deployment of a secure system.

The Three Mile Island nuclear accident is an appropriate example of the independence of the accreditation and the acquisition processes. Plant management's probabilistic risk assessment, with its event and fault trees, was ignored by operations personnel; but the assessment described quite precisely the scenarios that led to the radiation leakage. [9]

The Acquisition Life Cycle Does Not Properly Address Security

Security issues are rarely addressed in the second process, acquisition, illustrated in Table 2. There are but nine references to security and three references to threats in the eight key DoD

Table 2. The Acquisition Life Cycle Rarely Addresses Security

MAJOR PRODUCTS, REVIEWS, AUDITS, AND REFERENCES TO SECURITY	
Conceptual	Mission-Need Statement - Discuss Threats
	System Concept Paper - Describe Threats
	Operational Concept Document - No Security
	Other Conceptual Studies - No Security
	Test and Evaluation Master Plan - No Security
	System Engineering Management Plan - No Security
	System Requirements Review - No Security
Definition	Decision Coordinating Paper - Describe Threats
	System/Segment Specification - Specify Security Design and Compromise Requirements
	Interface Requirement Specification - No Security
	Software Development Plan - Specify Plan for Implementing Security Requirements
	Configuration Management - No Security
	System Design Review - Optional Review of Security Management
	Software Specification Review - No Security
Development	Development Specifications - No Security
	Product Specifications - No Security
	Software Quality Assurance - No Security
	Preliminary Design Review - Review Software Security
	Critical Design Review - Review Hardware Security Design
	Test Readiness Review - No Security
	Functional Configuration Audit - No Security
Test	Development Test and Evaluation Results - No Security
	Operational Test and Evaluation Results - No Security
	Physical Configuration Audit - No Security
	Formal Qualification Review - No Security
Operation	Engineering Change Reviews - No Security

directives, instructions, and implementing standards. Most are inadequate and do little to force the Government and development contractors to properly address security during system definition and development. Exceptions are the two recent Data Item Descriptions (DIDs) for the System/Segment Specification and the Software Development Plan which were released with DoD-STD-2167A, Defense System Software Development. [10, 11, 12]

Major DoD Policy Directives Do Not Mention Security

DoDD 5000.1 and DoDD 5000.3, Defense Acquisition Programs and Test and Evaluation, do nct address security. DoDI 5000.2, Major System Acquisition Procedures, addresses only Defense Intelligence Agency (DIA) validated threat discussions and threat descriptions required for three documents: the Mission-Need Statement (MNS), the System Concept Paper (SCP), and the Decision Coordinating Paper (DCP). There is no reference to security in the Test and Evaluation Master Plan (TEMP). [13, 14, 15]

Key DoD and Military Standards Scarcely Address Security

The system engineering standard, MIL-STD-499A, does not refer to security. There is no reference to security in the sections devoted to program planning and control, the System Engineering Management Plan (SEMP), and the acquisition process. [16]

In the software engineering standard, DoD-STD-2167A, security is mentioned: "the contractor shall comply with the security requirements specified in the contract" and "project characteristics may include security considerations in the operational environment." The DID for the System/Segment Specification states that one must "specify security requirements basic to the design of the system and security requirements necessary to prevent the compromise of sensitive information or materials." The DID for the Software Development Plan states that one "shall describe the contractor's plans for implementing the security requirements of the contract." [10, 11, 12]

In the original version of DoD-STD-2167 there is a DID for the Operational Concept Document (OCD). The purpose of the OCD is to "represent a consensus among development, support, and user agencies, and contractors on the operational concept of the system system being developed." The OCD does not mention security. [17]

The software quality assurance standard, DoD-STD-2168, does not refer to security. [18]

The system, software, and hardware review and audit standard, MIL-STD-1521B, addresses security only in relation to three reviews and audits. Security is cited as an example of a system engineering management activity that might be reviewed at the System Design Review; software security is identified as a Preliminary Design Review item ("an identification of unique security requirements and a description of the techniques to be used for implementing and maintaining security within the Computer Software Configuration Item shall be provided"); and the hardware detailed security engineering design is cited as a Critical Design Review item. [19]

In the specification standard, MIL-STD-490A, security markings are mentioned: "Specifications containing classified information shall be marked and handled in accordance with current security regulations as specified in the DoD 5220.22-M." [20 and 21]

Confusion for the System Developer

In general, the first process, accreditation, is independent - not tied to system definition, development, and test. Thus, the results of the risk assessments, ST&E, contingency planning, and other accreditation activities, see Table 1, do not influence definition of the system's functional baseline or influence subsequent development and product specifications. In the second process, acquisition, security requirements are not collectively addressed and are barely mentioned in relation to acquisition life cycle activities and the associated documentation.

The result is confusion for the system developer, and as a consequence a "bottom-up," or by-regulations implementation of security (involving, e.g., the use of Orange Book certified software from the Evaluated Products List, certified hardware from the Preferred Products List, cryptographic equipment from the Endorsed Cryptographic Products List, and shielded enclosures in accordance with NACSEM 5204) rather than a "top-down," or by-requirements implementation. After deployment the same systems fail, not because they do not meet regulations but because they fail to meet unspecified operational requirements. [22, 23, 24]

The system developer should therfore appreciate the limitations of regulations:

1) Security regulations do not do the obvious. Computer Security (COMPUSEC) regulations address security classification and compartmentation and usually do not address security loss of integrity, security destruction, and security denial of service. Communication Security (COMSEC) regulations do not address the damage caused by the high altitude nuclear explosion electromagnetic pulse (EMP) effect on unprotected electronic components. TEMPEST regulations do not address audio frequency compromise below 1 kHz. [24]

2) Security regulations address only known threats. There are security regulations for known threats with known security countermeasures. There are few security countermeasures and almost no security regulations for the newer and more sophisticated attacks, such as Trojan horses, trap doors, viruses, hardware spoofs, and password grabbers. [25]

3) Security regulations degrade performance. Table 3 lists a few of the key security regulations and guidelines that drive the

Table 3. Key Security Regulations and Guidelines

AFR 205-16
AR 380-380
CSC-STD-003-05
DCID 1/16
DIAM 50-3
DIAM 50-4
DoD 5200.1-R
DoD 5200.28-M
DoD 5200.28-STD
DoD 5220.22-M
DoD 5220.22-R
DoDD 5200.28
EO 12356
FIBS PUB 31
FIBS PUB 87
NACSEM 5201
NACSEM 5204
NACSEM 7002
NACSI 4009
NACSI 5004
NACSI 5005
NACSIM 5203
OMB Circular A-71
OPNAVINST 5239.1A
NCSC-TG-005

development of secure systems. There are major human overheads in procedural and administrative controls as well as COMPUSEC and COMSEC overheads. The latter include memory, disk, and processing system resources and associated decrease of system response due to identification, authentication, audit, erasure of residue storage, etc. "A totally secure system cannot operate," states Donn B. Parker, a computer security expert at SRI International. [25]

4) Security regulations are not security requirements. In general, security regulations address only security classification and compartmentation. Also, security requirements differ as a function of a system's operational environment. It is unlikely that any system has security requirements that correspond one-to-one with each of the 27 Trusted Computer Security Evaluation Criteria (TCSEC) for a given Orange Book division and class. For example, there are not the same discretionary access control, identification and authentication, and audit requirements for a guarded vault as there are for an aircraft or for an unmanned space platform. [22]

Integrate Security into the Acquisition Life Cycle

The solution to the problems reviewed is to integrate accreditation activities into the acquisition process and to ensure that all security requirements are specified in the functional baseline for design and test. This solution forces security requirements, like any other set of system requirements, to be collectively treated "top-down" and addressed in each phase of the acquisition life cycle - concept exploration through operations support.

AFR 205-16 contains a representative example of accreditation activities that can contribute directly to the realization of acquisition objectives, see Table 4. Such activities must always be tailored to system and security requirements and the appropriate level of trust. For higher levels, there are Orange Book and such other security assurance activities as those listed in Table 5. [22]

In time, enhancements reflecting this philosophy will be made. Accreditors will become more involved with acquisition, and system developers will follow more meaningful, DoD directives,

Table 4. AFR 205-16 Accreditation Activities

PHASE	ACTIONS REQUIRED
Conceptual	Identify and define general security requirement . Perform sensitivity assessment. Initiate risk assessment. Initiate economic assessment. Define functional security requirements including accuracy and validity. Translate functional security requirements to technical requirements. Approve functional security requirements. Identify required security management actions, for example, required certifications and approval. Develop detailed plans to satisfy security requirements. Review all aspects of security. Develop a security requirements baseline.
Definition	Develop detailed system or subsystem security specifications. Review and update sensitivity, risk, and economic assessments. Address results of risk analysis in design of security measures. Include ST&E concepts and plans. Review risk analysis. Review and approve security specifications as part of the overall specifications. Review ST&E plans.
Development	Review security specifications, ST&E plans, and security aspects of Operator and User Manuals. Develop and test security measures. Review program development to ensure compliance with security specifications. Update and approve ST&E evaluation plans.
Test	Test security measures. Complete ST&E. Certify that the system adequately addresses security requirements. Approve for operational use.
Operation	Consider security impact of all changes. Consider necessity and sufficiency of the existing security measures. Modify system to maintain cost-effective security. Reaccomplish risk analysis. Reapprove system.

Table 5. Additional Security Assurance Activities

Clandestine Vulnerability Analysis
Formal and Informal Security Policy Models
Formal Top and Low Level Specifications
Formal Verification
Covert Channel Analysis
Penetration Analysis
Trusted Facility Management
Security Configuration Management
Trusted Recovery
Trusted Distribution

instructions, and regulations.

There Are Four Major Benefits

1) Security requirements become part of a functional baseline. A systems approach is taken to the definition of a security requirements baseline. The security threats and the preliminary security operational requirements are specified in the MNS, SCP, and DCP. As appropriate, security threat and security operational requirement trade-offs are made in the sensitivity, risk, and economic assessments. The security operations concept and the security operational requirements are specified in the Operational Concept Document.

All security requirements including security operational requirements, compliant regulations, and approved waivers are specified in the System/Segment Specification and, if appropriate, in the Interface Requirements Specification. These documents become the functional baseline for subsequent Formal Qualification Review (FQR), and Development and Operational Test and Evaluation (DT&E and OT&E).

2) A systems approach is taken to secure development. Security is implemented by a single process. Accreditation, security development, and security test activities are addressed in the SEMP and TEMP. Security requirements are imposed and flow-down to all developmental and test activities, and are specifically addressed in the software requirement specifications, hardware development specifications, product specifications, software test documentation, and system test documentation.

3) Accreditation activities contribute to the acquisition life cycle activities. Accreditation is no longer an independent, merely document-producing process. System development activities are integrated. The results from risk assessment, contingency planning and test, ST&E, and all other accreditation activities contribute to acquisition life cycle activities.

4) Security requirements are met. Security requirements, security design, and security development are verified from phase to phase of the acquisition life cycle by established acquisition procedures that include requirement traceability activities, quality assurance activities, and formal design reviews. In a similar manner, the formal test reviews and the activities normally conducted toward the end of the acquisition life cycle ensure that security r quirements have been met in the development phase. Such validation includes FQRs of configurations items and the various DT&E and OT&E acceptance tests.

Historical Footnote

Confusion concerning security requirements is not new. In World War II an appropriate lesson was learned. Let us try now not to forget it. Anti-aircraft guns were placed on Liberty ships as a security measure, to counter the threat of hostile aircraft. Several months later, the guns on Liberty ships in the Mediterranean were removed because they did not meet the War Department's requirement for the destruction of enemy aircraft. Shortly after, several ships were sunk by hostile aircraft. Someone forgot the real requirement for anti-aircraft guns. [26]

References

[1] "'Very Serious Losses'; as the Navy Spy Scandel Spreads, Officials Assess the Damage," Time, June 17, 1985.

[2] "Strong and Silent - Can a new U.S. Sub Compete with Moscow's Best," Newsweek, September 12, 1988.

[3] "High-Tech Horror; How a $600 Million System Figured in a Ghastly Accident," Time, July 18, 1988.

[4] "Hard Sell: in Defense of the Exocet," Time, September 24, 1984.

[5] "Security Requirements for Automated Information Systems (AISs)," DoDD 5200.28, March 21, 1988.

[6] "Department of the Air Force Automatic Data Processing (ADP) Security Policy, Procedures, and Responsibilties," AFR 205-16, August 1, 1984.

[7] "Department of the Army Automation Security," AR 380-380, April 13, 1987.

[8] "Department of the Navy Automatic Data Processing Security Program," OPNAVINST 5239.1A, August 1, 1983.

[9] S. Levin, "Probabilistic Risk Assessment: Identifying the Real Risk of Nuclear Power," Technology Review, February-March 1984.

[10] "System/Segment Specification," DI-CMAN-80008A, February 29, 1988.

[11] "Software Development Plan," DI-MCCR-80030A, February 29, 1988.

[12] "Defense System Software Development," DoD Standard 2167A, February 29, 1988.

[13] "Major and Non-Major Defense Acquisition Program," DoDD 5000.1, September 23, 1987.

[14] "Defense Acquisition Program Procedures," DoDI 5000.2, September 1, 1987.

[15] "Test and Evaluation," DoDD 5000.3, March 12, 1986.

[16] "Engineering Management," MIL-STD-499A, May 1, 1974.

[17] "Operational Concept Document," DI-MCCR-80023, June 4, 1985.

[18] "Defense System Software Quality Program," DoD Standard 2168, April 29, 1988.

[19] "Technical Reviews and Audits for Systems, Equipments, and Computer Software," MIL-STD-1521B, December 19, 1985.

[20] "Specification Practices," MIL-STD-490A, June 4, 1985.

[21] "Industrial Security Manual for Safguarding Classified Information," DoD 5220.22-M, February 1985.

[22] "DoD Trusted Computer System Evaluation Criteria" DoD 5200.28-STD, December 26, 1985.

[23] "Information Security Products and Services Catalogue," NSA Quarterly (January, April, July, and October).

[24] "RF Shielded Enclosures for Communications Equipment," NACSEM 5204, Appendix B, October 30, 1964.

[25] "Is your Computer Infected? Systems fall to silent and contagious killers," Newsweek, February 1, 1988.

[26] "Measures of Effectiveness," SSM665 System Analysis Lecture Notes, USC, November 1979-January 1980.

Security Assurance through System Management

David Juitt
Secure Systems
Digital Equipment Corporation
85 Swanson Road
Boxborough, MA 01719-1326
Internet: juitt@ultra.enet.dec.com

Abstract

This paper details an ongoing advanced development effort within Digital Equipment Corporation to study security aspects of computing across a world-wide distributed environment and how they relate to conducting business safely. Recent results of the project are a security standard that was implemented throughout Digital's international computer network and a toolset for maintaining compliance with that standard. The toolset assists in the management of security-sensitive issues and provides a framework for delivering extended security management solutions in the future.

Introduction

Computer security in the commercial area is a widely discussed topic these days as companies strive to provide a safe method of doing business. As computing systems change, and corporations begin to take greater advantage of the latest advances in distributed system technology, a need exists to provide a more secure distributed environment. Efforts are under way at Digital to identify security-related issues that may affect methods of operation in our own distributed network, which consists of greater than 35,000 nodes located throughout the world.

Throughout both industry and academia, the past year has brought a rash of incidents that have underscored the inherent security vulnerabilities of large distributed networks. These incidents have brought the issue to light that conducting business over distributed networks often presents unanticipated risks that can manifest themselves in some future loss. These risks can often be addressed through an organizational standard for computer security. Without this formal standard, complex security decisions are inevitably left up to the individual, which often results in vulnerabilities open to exploitation.

The Need for a Security Standard

There is a need to define what computer security means to an organization. A corporate decision to rely on information systems as a way of conducting business implies the need for providing a corresponding security standard that covers those resources. The intent of such a security standard is to identify types of company data and information that must be protected, the degree of protection required, and the

The following are trademarks of Digital Equipment Corporation:
VMS, ULTRIX, DECNET, DECwindows, VAXcluster.

systems on which the data and information are located. The standard should help to achieve an equilibrium between providing access and capabilities to users and tightly controlling usage of security-sensitive information. This standard presents the definition of what security is in an organization.

The effort to develop a comprehensive security standard is as important as the efforts spent achieving the operational goals of the organization. It is critical that the security standard cover all environments and operations that are available in the distributed computing facility. Only after a standard is in place can the introduction and maintenance of compliance be effective.

Digital initiated an effort to collect information and define a computer security standard during the Spring of 1988. The effort involved a large number of participants from widely diversified areas of the organization. Over a period of about six months, input was collected and reviewed by a policy task force. The resulting document stood as the standard for computer security in Digital. The standard was then mapped into operating system terminology with a number of procedures defined that would allow a system manager to implement the policy.

The Need for Compliance

Once a codification of security management techniques was assembled, the design of a method to check compliance became possible. The actions necessary to ensure that the standard was being adhered to needed to be clearly defined. One requirement was to analyze each specific operating system and identify the security-relevant issues of system management, then to isolate the set of security-related functions. During the course of defining those actions and needs, it became obvious that in some cases day-to-day attention was needed. It also became apparent that a standard set of tools was needed for continual security standard maintenance.

Operating systems such as VMS and ULTRIX contain many system features and parameters that, if set and maintained properly, can offer considerable resistance to security threats. System managers may be unaware of these features or the role that these features play in security. Requiring a system manager to learn many different interfaces greatly complicates what is often a new task.

Close examination showed the need for network security was increasingly addressed by good system security. The very process of securing individual nodes, which evolve into large networks, has helped begin to merge the past disjoint disciplines of network and system security. When this method is extended, along with the issue of securing network traffic, control of our distributed computing resources can be maintained. This examination and analysis of outstanding needs shifted the project toward attempting to develop a locally based solution to the problem of security standard compliance.

Development of a Toolset

After the security standard was defined, and its impact understood, we designed a toolset to ease the system managers' job of maintaining compliance. It was decided that an easy-to-use, highly reliable solution was needed by the already overworked system managers. Some of the goals of the toolset were:

- Consistency of use across Digital's internationally distributed computer network.
- Implementation in a high-level programming language. Past experience had shown that command scripts could be modified too easily, thus reducing the integrity of the results.
- A centralized reporting feature to address the problem of auditing compliance. In this manner, the tools could provide information that would ensure that all nodes in the network were conforming to the security standard.
- A remote testing feature that would allow for quick placement of future enhancements to security testing.

A software toolset was developed that comprised an extensible framework, a security daemon and an adaptable interface capable of greatly assisting a system manager in the task of being a security manager. The security tests performed are defined by the Digital security standard. The framework permits locally developed tests to be easily integrated into local procedures. The toolset provides a clearly defined, two-level reporting system for compliance auditing from the viewpoint of a network manager. The reports generated can be used locally to manage the system, or a separate condensed copy of the results can be centrally collected. The problem of maintaining the integrity of the collected data was addressed.

At the highest level the toolset provides functions to perform:

- Scheduling
- Execution of tests
- Selected lockdown and
- Reporting

The scheduling functions are provided by the security daemon. The daemon spawns processes that are dispatched into the system to perform a single security test. The tests that are executed by the toolset examine the attributes of five major subsystems within an operating system:

- File System

  The File System tasks are geared toward detecting situations when critical system files may not be adequately protected. The results of the tests may require changes restricting the access to system directories. These changes should be transparent to ordinary users. The tests also require users to limit the openness of their own directories by storing publicly accessible files in public or project libraries or in specially created and protected user directories.

- Accounts

  The purpose of the Account subsystem is to ensure that: (1) only bona fide employees and contract personnel are given accounts and (2) the level of access provided matches their job responsibilities. These checks, along with good password management (password length and expiration time) can reduce the vulnerability of systems to penetration.

- Network

  The Digital network has traditionally configured its network objects to provide relatively open access. The Network subsystem requires the elimination of nonessential network objects and the restriction of others to controlled use.

- Startup

  The system generation security parameters available in VMS define the thresholds at which a given system will decide: (1) that a breakin attempt is under way and (2) the nature of the evasive action to take. The Startup subsystem checks for minimal levels of these and related security parameters.

- Security Auditing

  Auditing is a critical element in the plan for a secure environment. Only if there are regular reports of significant security-related events, and these reports are reviewed by system owners and managers, can possible penetration attempts be detected and investigated. This subsystem ensures that the correct set of events are enabled to be audited.

Selecting Tests

The menu-driven, DECwindows compatible, user interface allows the system manager to establish one or more groups of tasks to check the security of the system. These tasks are assigned to named groups that create a team of system security management agents. The granularity of testing can be assigned in any or all of the following ways:

- A single autonomous task in a system
- A group of such tasks, spread across subsystems
- One or more complete, independent subsystems
- The full complement of subsystem analyses

The results of each group of initiated tests are stored in a local database file. This file is also used for the synchronization required by groups of tests that are executed in a VAXcluster environment.

Once the toolset detects a violation of the security standard a report is generated detailing the area of non-compliance. If desired by the system manager, a command procedure can also be generated containing the commands to adjust system parameters automatically such that compliance with the established security standard is enforced. These command procedures are known as lockdown files, named after the function they perform.

Report collection is performed after all tests complete execution. The daemon assembles a complete report from the individual results of each test. Test processes are responsible for returning status codes and storing report fragments in files. The daemon then mails a copy of the full report to the system manager. A second level of reporting is available for use by a network manager to collect statistical data regarding compliance throughout the network.

Future Directions

The toolset as presented above represents an extensible framework on which additional security solutions can be based. Tools that handle alarm interpretation and intrusion detection are future possibilities. Image authentication management to prevent the unauthorized modification of system images is another area that can be explored. There is still a need to address the difficult security problem associated with the misuse of privilege that is commonly caused by uninformed users.

Accomplishments

This project was able to help identify and isolate the areas of system management that are security sensitive. It has also illustrated the importance of developing a security standard before the implementation of security tools. The security standard definition and toolset framework presented in this paper allow for new extensions to take advantage of additional security-enhancing features of operating systems and network software as they become available.

A SYSTEMATIC APPROACH TO SOFTWARE SECURITY EVALUATIONS*

Mary Frances Theofanos
Data Systems Research and Development
Martin Marietta Energy Systems, Inc.
P.O. Box 2003
Oak Ridge, TN 37831-7346

Abstract

A security certification of the word-processing and communications software to be used on the U.S. Department of State's computer systems was carried out. A novel approach to computer security evaluation and testing based on National Institute of Standards and Technology criteria was used. Software controls were evaluated and points of vulnerability were identified. This test was of a stand-alone system but could be applied to all levels of computing.

Introduction

Background

The issues of computer–system security and security evaluations have become increasingly important in recent years, particularly for government systems containing sensitive data. [1, 2] The Joint Center for Information Security Technology (JCIST), located at the Oak Ridge National Laboratory (ORNL), addresses some of these issues as part of its mission. JCIST recently performed a security certification of the software to be used on Wang VS-series computers in the U.S. Department of State's (DOS's) Foreign Affairs Information System (FAIS) Early Operational Capability (EOC). This certification was arranged through an interagency agreement between DOS and the U.S. Department of Energy (DOE). The purpose of this certification was to determine the extent to which FAIS software complies with the security requirements mandated by DOS when it is used with specified versions of the vendor operating system, utilities, and application software. This paper describes the methodology and approach used by JCIST to conduct this certification.

The security certification effort was composed of three parts (basic, detailed, and penetration testing) as outlined in Guidelines For Computer Security Certification and Accreditation. [1] The basic evaluation examined the system and identified inherent security exposures and controls. This phase focused on the system design to determine whether it was complete. The basic evaluation set the stage for detailed evaluation, in which the adequacy of the controls was tested on an actual system. The detailed evaluation systematically tested the coverage of each exposure by its associated controls.

Both the basic and the detailed evaluations focused on the activities performed by a legitimate FAIS user exercising the FAIS functions and other parts of the Wang system in normal ways. Vulnerability to attacks by outsiders or attempts by FAIS users to circumvent the system's security in nontraditional ways were addressed by penetration testing. Penetration tests verified the adequacy of system controls for protecting security features within the system and determined whether the system is immune to violation from the outside.

* The submitted manuscript, prepared for the U.S. Department of State, has been authored by a contractor of the U.S. Government, under U.S. Department of Energy contract DE-AC05 84OR21400 with Martin Marietta Energy Systems Inc., accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

This approach was effective for two reasons. First, the system was reduced to a set of conceptually smaller systems, each of which performed one function. As the the analysis progressed from general to detailed, the analyst considered smaller, more understandable parts of the system. Second, the context of each function was readily apparent, giving the analyst a framework within which to judge vulnerability at each point in the system.

Overview of FAIS

The 10-year FAIS program is the strategy adopted by DOS to implement a computer architecture that integrates office automation, data processing and telecommunications for its offices throughout the world. Thus, FAIS is the DOS architecture for classified information processing. EOC is the first implementation of individual elements of FAIS in the live DOS classified environment and represents an operational test of both the system concept and the FAIS software suite. The EOC phase uses existing DOS computers and off-the-shelf software when possible; customized software provides additional functionality.

Most DOS personnel did not have direct automated capabilities to send and receive formal or informal mail from their workstations. Access to the data bases containing official documents was also limited. FAIS provides electronic capabilities to most users who were previously isolated on stand-alone minicomputers and office automation systems. Services provided by the system include word processing, local filing and retrieval, interorganizational electronic mail, communications, logon to DOS data bases, a telegraphic interface, and security features.

Evaluation Methodologies

Basic Evaluation

The basic evaluation methodology applied to FAIS EOC was first proposed by Pfleeger and Pfleeger. [2] This section briefly describes each of the steps that were taken by JCIST in applying the basic evaluation methodology. The purpose of this analysis was to verify the existence of controls and to rate each control according to its potential effectiveness.

The basic evaluation examined the FAIS EOC system design documentation to determine if effective controls were included to prevent exploitation of security exposures. This evaluation methodically determined the potential exposures, associated controls with exposures, and assessed the effectiveness of the controls. The FAIS system is is used primarily for preparation and transmission of official documents. Therefore the basic evaluation emphasizes documents found in the system and the transactions performed on them. The basic evaluation was performed using only the system documentation to determine whether (1) at least one control was available for every identified exposure and (2) the controls, as designed, were adequate to prevent exploitation of the exposure.

Determination of Transaction Flows. The first step was to determine the types of data flowing through the system. A complete list of data items was generated from the system documentation. Several assumptions made it possible to reduce this large list to nine data object types without compromising the security analysis. The basic evaluation considered the following nine major data types: telegram, informal message, action memorandum, briefing memorandum, information memorandum, memorandum (other), issue paper, envelope, and WP Plus document.

The next step was to define all possible transactions or operations that could be performed on the data objects. The following transactions were identified: create, revise, clear and approve, view, print, copy, transmit, delete, and determine the existence of. A transaction flow diagram was drawn for each combination of data object and transaction, depicting the movement of the data object through all modules within the system that could affect it (Fig. 1).

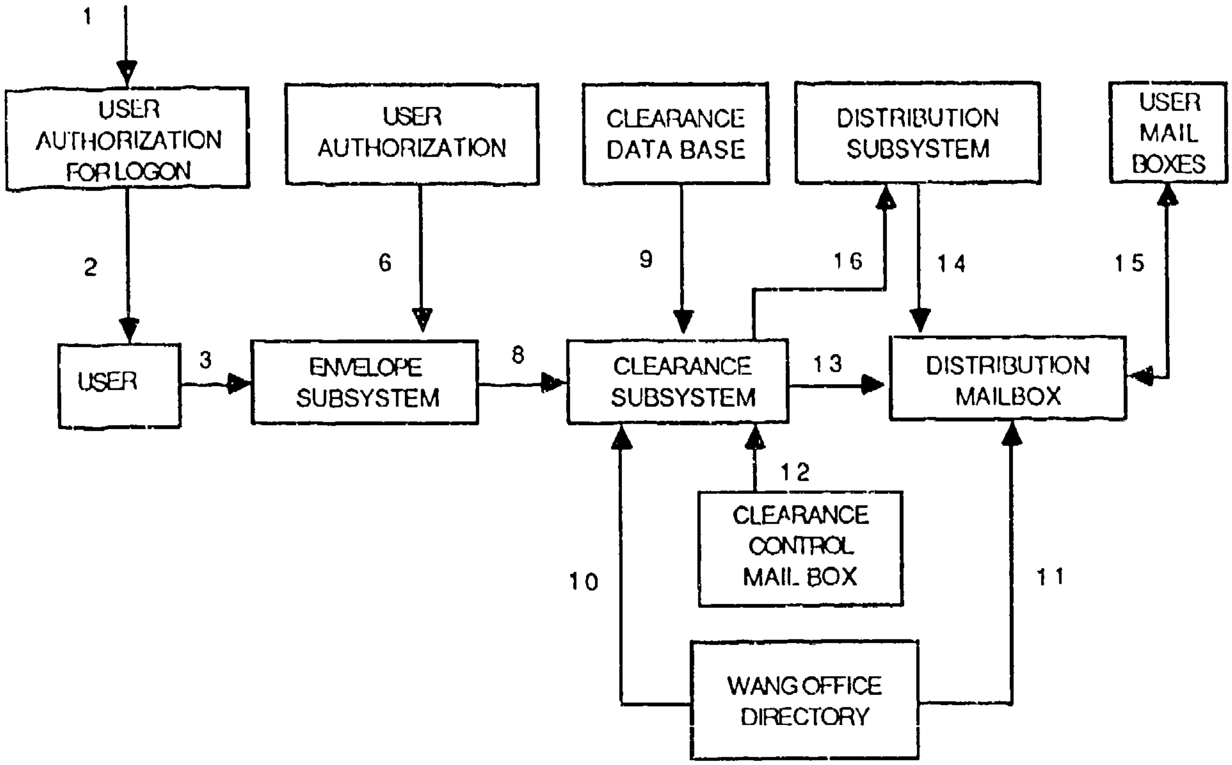


Figure 1. Telegram Clearance and Approval Transaction Flow.

Evaluation of Transaction Flow for Exposures. The types of vulnerabilities in the system were derived from the EOC security requirements as defined by DOS. [3] According to the requirements document, the system security requirements for FAIS EOC can be categorized as follows: discretionary access controls, use of internal and external labels, accountability, auditability, and continuous protection. Forty-one detailed requirements were identified for EOC. These requirements were derived and grouped according to the minimum requirements listed above. Sixty-two potential system exposures was identified from the system security requirements. These exposures were also grouped into the general categories they address.

To provide a systematic approach to identifying the vulnerable points in the system, potential exposures were associated with each step of the transaction flow diagrams. A new table was generated from this mapping that shows each step of every transaction flow and every exposure that might occur at that step. This mapping was meaningful, but a measure was required to distinguish the seriousness of individual exposures. To determine this likelihood of exploitation, the knowledge and skills required to take advantage of an exposure were considered. Each exposure was evaluated and assigned one of the following six ratings denoting the likelihood of occurrence:

D1 essentially impossible to exploit,
D2 a system administrator or security officer could exploit,
D3 an operator could exploit,
D4 a system user with inside information could exploit,
D5 any system user could exploit, and
D6 any person could exploit.

The severity of an exposure is defined as the most serious (highest numbered) applicable rating from this list. Upon completion of this step, a table was generated indicating the rating for each of the 62 exposures.

Identifying Controls. FAIS contains a set of security safeguards (called "controls") designed to reduce exposures. The system documentation was studied and 114 controls were identified and enumerated in a table. A "stringency" measure, designed to focus on human oversight and/or intentional misuse, was measured by determining who could decide whether to use the control or not. Each of the controls was assigned a stringency label to indicate the effectiveness of that particular control:

S1 cannot be avoided,
S2 system control, at the discretion of the security administrator,
S3 system control, at the discretion of the operator,
S4 system control, at the discretion of the user, and
S5 procedural or administrative control.

Mapping Exposures and Controls. In most cases, an exposure was covered by multiple controls. This set of controls included those designed to prevent exploitation of an exposure as well as those designed to detect exploitation after the occurrence. This step produced a table of a one to many mapping of exposures to controls.

Evaluating Control Adequacy. The last step of the basic evaluation was to determine whether the controls designed into the FAIS system were adequate to meet the specified security requirements and general security needs of the system. Every exposure and its corresponding set of controls was examined to determine whether the set of controls was capable of preventing or detecting any exploitation of that exposure. The effectiveness of each set of controls was categorized as one of the following:

(A) able to prevent an occurrence,
(B) able to detect an occurrence only after the fact, or
(C) able neither to prevent nor detect an occurrence.

A table listing the exposures, the difficulty of exploiting each exposure, and the effectiveness of the set of controls in preventing that exposure from being exploited was generated as the final step of the basic evaluation.

Data Base. The tables and lists generated during the basic evaluation were collected in a data base. [4] The data base provided a convenient way to store, organize, and manipulate the large amount of data identified and collected during the basic evaluation. It was an important tool in assisting the methodical approach to testing the actual system. The data base was developed from the low level analyses of the basic evaluation that are relatively easy to understand to support the more comprehensive detailed evaluation.

Detailed Evaluation

The purpose of the detailed evaluation is to test the adequacy of the controls on an actual system. The detailed evaluation proceeds systematically to test each exposure at each step of each transaction that can be performed on each object. Exhaustive application of this approach should locate all instances where a user performing normal system functions can violate the security requirements of the system.

Potential exposures present at each step of each transaction are examined. The data base contains all these step/exposure/control relationships and can automatically generate test scripts. The data base provides a systematic way of progressing through each possible path a user could take on the system. Thus, it is possible to print a list of all controls for all exposures for all steps. For the purposes of detailed evaluation, it is unimportant which control(s) prevented exploitation of an exposure. The detailed evaluation only seeks to establish that some control(s) prevents this exploitation and that therefore a security requirement has been satisfied.

Vulnerabilibty Reports. If system vulnerabilities are identified, the system tester completes a vulnerability report. On the report form, the tester identifies precisely what test was performed; what conditions were in effect at the time of the test (for example, under what user privileges the tester was operating); what results should have been obtained; and what results were actually wer obtained. The vulnerability reports are numbered for reference purposes.

Test Users. Legitimate FAIS users must be defined because the detailed evaluation focuses on the functions performed by legitimate FAIS users exercising the system. A system security administrator's (SSA) manual describes the proper configuration and control of the automated and administrative security controls.[5] The SSA manual lists five classes of users on the FAIS computer system (general user, operator, system manager, system security officer, and configuration manager). The detailed evaluation focuses primarily on the general user, who must be defined in five separate profiles on the system (VS, Wang, FAIS, AMU, and WP Plus).

Only two profiles require site-specific information if the SSA manual's guidelines are adhered to. According to the parameters of the FAIS and WP Plus profiles, at least 16 users must be defined to adequately represent the general user population. In the FAIS profile there are five access levels to approve and originate documents: Top Secret (TS), Secret (SE), Confidential (CO), Limited Official Use (LO), and Unclassified (UN). In our test system, there was at least one user for each combination of clearance levels. Representative channel and caption parameters were also assigned. Two classes of users were defined in the WP Plus profiles with differing library access to test the sharing of libraries. One user was given no access to WP Plus. In addition to these general users, the system also contains an SSA and an operator.

Detailed Evaluation Approach. The detailed testing was performed on a WANG VS 85 minicomputer, configured as a DOS bureau processor. The evaluation included processing on a stand-alone computer system only. This test did not evaluate any networking interfaces because the system was not connected to any other computer systems.

The detailed evaluation methodology describes the exhaustive testing of each exposure at each step of each transaction of every object on the system. The transaction flow diagrams generated during the basic evaluation show that many of these tests are redundant. As a result, the first object tested required extensive testing and evaluation. This test focused on all combinations of users and input and provided a baseline from which subsequent tests would be performed.

Given the thoroughness and completeness of the initial testing, the methodology allowed for the elimination of redundant tests by identifying the unique paths or elements of the subsequent tests. These unique paths required complete analysis and testing. Because of this approach, the number of tests was reduced without compromising the effectiveness or completeness of the detailed evaluation methodology.

Penetration Testing

Penetration testing attempts to identify errors that might not have been located through either the basic or the detailed evaluations. It attempts to identify security flaws that could be exploited by authorized users or by outsiders to circumvent or defeat the security of the system. The general penetration testing approach is based upon an analysis of the system, hypotheses of possible flaws, confirmation or rejection of hypotheses, and extension of confirmed hypotheses. [6]

The goals of the FAIS EOC evaluation were to assess the penetration resistance of FAIS software suites, help determine the difficulties involved in exploiting flaws, and provide a clear demonstration of FAIS flaw exploitability. The penetration testing approach involved three steps: identify sensitive objects, determine points of vulnerability, and test vulnerabilities to determine the adequacy of controls.

Identify Sensitive Objects. For penetration testing, "sensitive objects" are the data and program modules on which the security of the system depends; they implement the security of the system. Thus, to identify sensitive objects, it was necessary to study the control measures and devices that prevent exploitation of vulnerabilities in the security system. A sensitive object may be a collection of security objects. For instance, the "Wang user profile" includes several separate security objects such as a 3-character logon ID and a password. By reviewing the implementation of controls and focusing on representations of data items, the security objects of the FAIS system were combined into the sensitive objects. The sensitive objects were then categorized according to their source or the application that performed the various security functions.

Each control description involved at least one security object and its use; each security object is used in (or by) at least one control. An analysis of these controls produced a listing of about 70 security objects. These security objects were determined from the security features of the system.

Determine Points of Vulnerability. Points of vulnerability were determined in two steps: identification of security control functions and matrix development. Because sensitive objects are the controls that implement the security of the system, they are precisely the means whereby a perpetrator can exploit the security of the system. Hence, the protection of sensitive objects (security control functions) and the ways this protection can be circumvented were examined.

A sensitive function protects or implements a sensitive object. For example, the system userlist, a sensitive object, contains the IDs, names, passwords, and access rights of all designated system users. The security utility is used to add, delete, or modify records of the userlist and was therefore identified as a sensitive function.

A list of sensitive functions was constructed and steps were taken to verify the completeness of the list of sensitive functions. A complete list was produced of all transactions during a test that showed all files that were opened and all security functions accessed during that particular test. Unfortunately, the amount of additional output generated by the audit log prohibited the usefulness of this approach to verify every function. At the end of this phase, the information gathered was represented as a lattice structure (Fig. 2).

A matrix was constructed from the lattice structure. The matrix depicts the controls of the overall FAIS system and the functions that implement them. The columns of the matrix represent the sensitive objects of the system, and the rows represent the functions that access them. Thus each cell in the matrix indicates that a particular function interacts with a particular object. In addition, each nonblank entry attempts to categorize the type of interaction.

The matrix was the backbone of the testing approach. The functions and the effect they have on the associated objects could be examined methodically using this structure. Moreover, the scripts were written to target specific function/object pairs in an attempt to defeat the effectiveness of the controls they implement. To examine each function/object pair from a number of users' perspectives it was necessary to identify the general categories of users available on the system. Five classes of users, grouped according to their access privileges, were identified. These classes range from an illegal user who has no authorized access privileges to the system to an FAIS super user who has complete access to the entire system.

When tests were completed for each nonempty cell of the matrix, all points of direct attack had been tested.

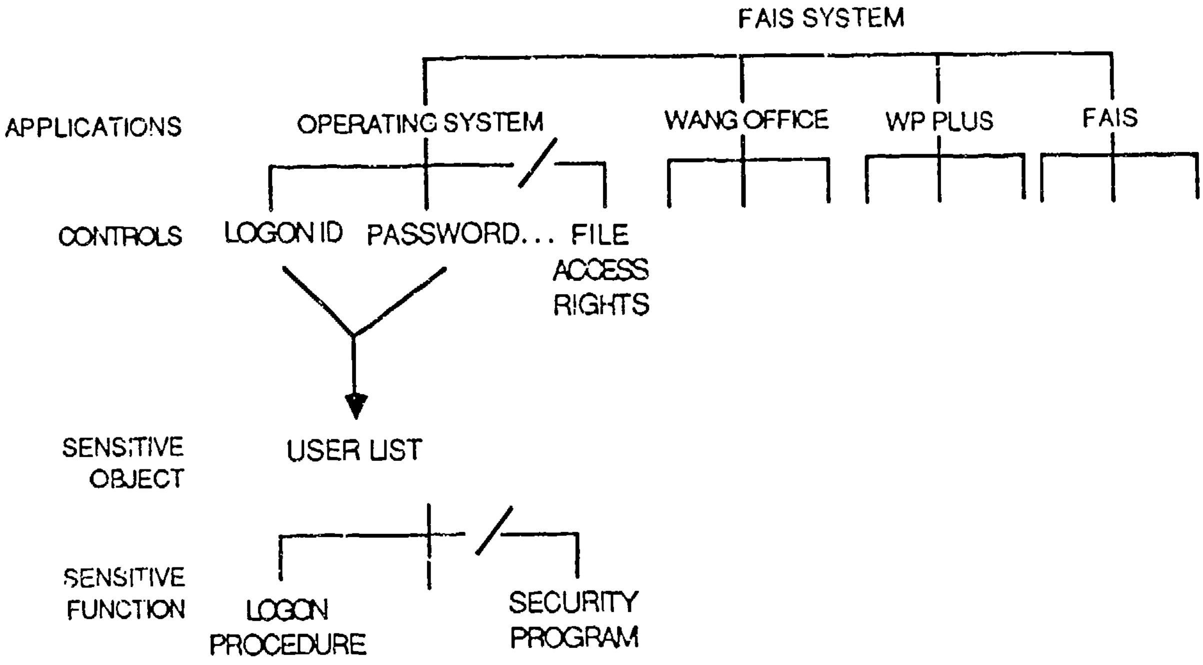


Figure 2. Lattice structure of the FAIS system has controls accessing sensitive objects protected and implemented by security control functions.

A sensitive object can be compromised in one of two ways: by direct attack (effective 'or a specific object) or an indirect, "backdoor" attack (effective on objects of the given type). Direct penetration testing focuses on the integrity of the specified goals of the protection scheme. Indirect attacks attempt to target properties and characteristics of the system hardware and software that are unknown or little known to the analyst. They include deliberate and malicious attempts to compromise a sensitive object and may be perpetrated by both authorized and unauthorized users. Unlike conventional direct attacks, indirect attacks exploit vulnerabilities that are not based upon errors or omissions in the security software. Methods employed in such attacks fall into two general categories: those that use undocumented functions or procedures in software or hardwa'e, and inventive application of system components in a novel or unorthodox manner. Penetration testing is complete only when all forms of these two types of attacks have been checked.

Test for Vulnerabilities. The objects and functions in the system were partitioned among the security analysts. Following the test scripts associated with the object/function pairs, each analyst tried to undermine the effectiveness of the given function or object. The results of each test were recorded. If unauthorized access to an object was discovered, a vulnerability report form (identical to those used in the detailed evaluation) was completed to describe the testing and the results.

The results were analyzed after each phase of testing. Often this analysis suggested further tests that should be performed. Analysis of the results from these new tests continued the cycle of testing, analysis, and generation of new tests. The iterative process terminated when the analysis indicated that further testing was unlikely to reveal additional weaknesses.

Evaluation Results

Basic Evaluation

Each requirement is associated with a set of controls, but the basic evaluation looked beyond requirements to potential exposures and the resulting weaknesses in the system design. To complete the evaluation, the tables developed during each step of the basic evaluation were analyzed to identify the weakest points in the system. The difficulty of exploiting each exposure was assessed. There were six ratings for the likelihood of occurrence for exposures ranging from "impossible to exploit"(D1) to "any person could exploit"(D6). None of the 62 potential exposures were assigned a D6 rating, indicating that any person could exploit the exposure; 10 exposures were assigned a D5 rating level, suggesting that any system user could exploit them.

The analysis of the table mapping exposures [illegible] ets of controls also indicated that each potential exposure is covered by at least one cont. . ., and most potential exposures are addressed by a set of controls. Each identified control was assigned a stringency rating. The weakest controls in the system were those rated S5 (administrative controls) or those at the discretion of the user, rated S4. Twenty-two of the 114 controls were rated as S5, or administrative controls.

The final step of the basic evaluation determined if the set of controls was capable of preventing or detecting any exploitation of the exposures. In this step, each set of controls was categorized according to its effectiveness. The lowest rating of C was assigned to those controls able neither to prevent nor detect an occurrence of the exploitation of that exposure. Of the 62 exposures and their corresponding controls, only 9 were given a C rating.

The nine weak exposure/control sets were compared with the ten most exploitable exposures. Only three exposures appear in both lists. Thus three areas were identified where weak controls combine with ease of potential exploitation. According to the basic evaluation, these areas are considered to be the points at which the system is most vulnerable.

Detailed Evaluation

The detailed evaluation does not attempt to assess the potential impact of the shortcomings or the difficulty of eliminating them. It proceeded systematically to test the adequacy of the controls on an actual system and compared the security requirements specification with the functions of the operational FAIS system. When comparing the security requirements with the implemented system to determine if the requirements were satisfied, a strict interpretation was applied to each security requirement. For instance, if the requirement specified that every object stored within a system must be marked with a label identifying the sensitivity of the object, and an object was discovered that was not labeled appropriately, then the requirement was not satisfied.

The evaluation showed that 22 of the 41 security requirements specified in the security requirements document have not been satisfied for EOC. Of the 22 requirements, 6 refer to access control problems. Six more of the requirements fail due to labeling of objects, especially informal messages. Two of the accountability requirements were not satisfied. The audit log testing determined that the audit log fails to meet four of the auditability requirements. Finally, four of the requirements designed for continuous protection were not satisfied.

The results of the detailed evaluation were not a surprise. It was anticipated that a large number of the requirements would fail given the strict interpretation applied. DOS viewed this analysis as a baseline or benchmark. DOS plans to assess the impact of these vulnerabilities on current operations. More importantly, DOS plans to use this information to identify corrective measures, to evaluate its current security requirements and needs, and to resolve security issues to improve the security of future systems.

Penetration Testing

The object of the penetration effort was not to find all the flaws in the system, but to provide an assessment of the application's penetration resistance. As was the case with the detailed evaluation, the penetration effort does not attempt to assess the potential impact of these vulnerabilities or the difficulty of eliminating them.

The penetration testing and evaluation generated nine vulnerability reports, each of which documents a specific security failure. Each of these vulnerabilities is described in detail in these reports. The actual vulnerability, the steps and circumstances necessary to exploit the security failure, as well as an indication of the type of user that may take advantage of the situation are included in these descriptions.

This report cannot provide specific information describing the actual vulnerabilities discovered. The penetration evaluation did provide a demonstration of the exploitability of five vulnerable areas. The vulnerabilities included the ability of an authorized user to gain access to additional commands and files, acquire additional user privileges, and bypass envelope security. This evaluation did not demonstrate the ability of an illegitimate user to penetrate the system, but did show that authorized users can expand their access.

Conclusion

In this paper, we have described a novel approach to computer security evaluation and testing of an existing system. Although it has been applied to a stand-alone system only, this approach can be taken to evaluate the security of computing systems at all levels. For example, the same methodology can be used to test a complex network of computers or individual applications such as data bases.

This approach has several advantages. First, the method is completely systematic and provides a well-defined sequence of steps leading from the basic evaluation, through the detailed evaluation, to penetiation testing. This systematic methodology minimizes the chance of overlooking a basic flaw in the security of the system.

A second advantage of this methodology makes the huge task of security evaluation manageable. Each step builds on the results of the pievious step. First, the basic evaluation determines whether the design of the security system (as described by the documentation) yields an appropriate level of security. The results of the basic evaluation are then used in the detailed evaluation, which systematically tests the actual system to determine whether the documentation accurately reflects the level of security of the system itself. Finally, the results of the detailed evaluation are used in the penetration testing phase, which systematically tests the adequacy of system controls for protecting the objects that provide security within the system. This methodology is one of few organized approaches to generating particular test cases for penetration testing.

A third major advantage of this approach is that security certification does not have to be carried out by a computer security expert. Testing at all levels can be carried out by a systems analyst familiar with the system under investigation. The methodology described in this paper can be applied widely because the availability of systems analysts is far greater than that of computer security experts.

Finally, the approach is based upon a recognized national standard for computer security evaluation. [1] Hence, the systems analyst can have confidence that he/she is using a security testing methodology whose approach was derived from a reference document written by the National Institute of Standards and Technology (formerly the National Bureau of Standards) for establishing and performing a certification program for computer security. This provides a degree of reassurance to any agency using this methodology to test the security of its computing systems.

This is only the first step in DOS's computer se urity evaluation process. This is but one of several certification efforts necessary in accrediting FAIS EOC for operation as a top secret computer network. The results of this study must be used in conjunction with the results of the verification and validation effort as well as the system and functional testing results of the system dev 'oper. Finally, the results of the FAIS EOC network security evaluation currently in progress must also be taken into account in order to obtain an overall picture of the security of the FAIS system.

References

[1] Guidelines for Computer Security Certification and Accreditation, Federal Information Processing Standards Publication 102, U.S. Department of Commerce, National Bureau of Standards, September 1983.

[2] C.P. Pfleeger and S.L. Pfleeger, "A Transaction Flow Approach to Computer Security Certification," Computers & Security, Vol. 7, pp. 495–502, Oct. 1988.

[3] Planning Research Corporation, Foreign Affairs Information System Early Operational Capability Security Requirements Allocation, McLean, VA., May 1986.

[4] C.P. Pfleeger, S.L. Pfleeger, and M.F. Theofanos, Data Base to Support a Transaction Flow Approach to Computer Security Certification, Martin Marietta Energy Systems Technical Report, K–DSRD 161.

[5] Planning Research Corporation, Foreign Affairs Information System Early Operational Capability ISSO Manual, McLean, VA., Sept. 1987.

[6] C.P. Pfleeger, S.L. Pfleeger, and M.F. Theofanos,"A Methodology For Penetration Testing," submitted to Computers & Security, March 1989.

Executive Summary

PROFESSIONAL CERTIFICATION FOR COMPUTER SECURITY PRACTITIONERS

Toni Fish & Sally Meglathery

International Information Systems Security Certification Consortium
(ISC)2
P.O. Box 98
Spencer, MA 01562-0098

Introduction

In 1986, the Information Systems Security Association (ISSA) began a process of creating a program for certification of professional practitioners in information systems security. In 1987 and 1988, the National Security Agency (NSA) sponsored two workshops, at the University of Maryland and at Idaho State University (ISU), that created a number of modules intended to be used by universities in teaching computer security in engineering and in business of MIS programs. In 1988, the Special Interest Group for Computer Security (SIG-CS) of the Data Processing Management Association (DPMA) was able to bring together a consortium including NSA, Idaho State University, DPMA, ISSA, and the Computer Security Institute, to propose the creation of a consortium to continue the development of such a professional certification program.

This discussion summarizes the planning work that has been done since early 1988 and outlines the plan that has been developed, which will lead to the first formal certification examination by September 1991. The work sponsored by NSA in the curriculum modules has been combined with work of ISSA in the consortium to serve as the foundation for a Common Body of Knowledge. ISU is serving to maintain a common data bank of information, bibliography, and soon test questions and answers with justifications. All of these elements will support the development of the certification program, and will serve as an important source of information in future maintenance of the certification examinations and program.

The International Information Systems Security Certification Consortium (ISC)^2 brings together representatives from many professional organizations in the information processing field, from academe, and from the United States and Canadian governments. This collection of talent and resources is producing a certification program and a data bank that will serve as a significant resource and for formalization of the profession for many years.

The Project

The (ISC)^2 has developed a document which outlines the events that are significant in this project, and it presents some information about the activities and people involved in each stage. This document also includes a complete list of the more detailed activities.

Briefly, the project of creating a certification program involves planning the work; identifying a Common Body of Knowledge; planning, creating, and validating an examination; and establishing committees to maintain the Common Body of Knowledge and examination and certification process as dynamic entities that mirror the changes in information systems technology and security concerns. The work is underway now; the first formal examination is scheduled for shortly after September 1991.

The Players

The players include representatives from the private sector, from volunteer professional associations, from the academic world, and from the United States and Canadian governments. The structure of (ISC)^2 permits inclusion of other interested bodies as the program develops. The major computerized information resource is maintained by Idaho State University to ensure its availability to all (ISC)^2 members and to support the certification program in the future. This resource is derived from wo:k sponsored by NSA and as it is augmented during our project the resource will grow in value to all security professionals.

Integrating Security Requirements and Software Development Standards

T.C. Vickers Benzel*
Trusted Information Systems Inc.
11340 W. Olympic Blvd. Suite 265
Los Angeles, CA 90064
213-477-5828

March 1989

*This work was supported by the Rome Air Development Center under contract F30602-86-C-0048.

1 Introduction

The Unisys Corporation and Trusted Information Systems, Inc. are analyzing the security requirements of complex battle management systems under a contract to the Rome Air Development Center. The tasks under this contract are recommending methodologies to address system architecture, development and eventual accreditation.

In order to provide assurances that a complex battle management system can satisfy its security requirements, it is necessary to examine security in the software development process in addition to examining security issues for the battle management system architecture, accreditation, and design. This report is the second in a series [4] of investigations on security requirements with respect to a software development methodology for complex battle management systems. The software development methodology is defined here to include the tools, techniques, and processes which when used in a structured manner can lead to a cohesive software product.

This paper reports on the results of our examination of the software development process when applied in conjunction with security requirements to a developing system. Section 2 discusses the need for integrating the software development process of DOD-STD-2167A and security requirements; it then provides a brief description of the software development process of DOD-STD-2167A and the Trusted Computer Systems Evaluation Criteria (TCSEC) development paradigm as background. Section 3 discusses several key issues with respect to integrating DOD-STD-2167A and the TCSEC. Section 4 proposes a first approximation at a tailored software development process integrating security requirements with DOD-STD-2167A requirements. The integrated approach relies on an iterative model of software development and recommends tailoring rather than revising DOD-STD-2167A. Section 5 presents conclusions and plans for future refinement of the integrated trusted software development process.

2 Need for Integration

Currently the DOD has one set of regulations governing the software development process and a separate set of regulations governing the development of trusted computing systems. The central regulation in the software development arena is the Military Standard: Defense System Software Development (DOD-STD-2167A) [13]. The central regulation in the trusted computing systems arena is the Trusted Computer Systems Evaluation Criteria (TCSEC) [5].

Historically, security requirements and software development standards have been treated separately. While the TCSEC (the most commonly used source of security requirements) embodies principles for developing trusted systems, it does not address the software development process. On the other hand, DOD-STD-2167A imposes explicit software development requirements and, in doing so, implies a specific process to follow for developing "good" software.

Experience has shown that retrofitting security requirements late in the software development process leads to systems which do not adequately satisfy security requirements [6]. Recent attempts at including security in the early phases of a project often result in a "two-track approach" to system development. That is, software developers proceed within the "traditional" approach to development while the security group conducts a parallel effort driven largely by the TCSEC security requirements. The two-track approach inevitably leads to one of the tracks dominating the process while the secondary track is virtually ignored. Because the use of a "traditional" software development approach, such as that embodied in DOD-STD-2167A, is often contractually mandated, it is the security development which suffers. In fact, when software development faces budget and schedule problems the security track is often greatly reduced. Even under the best of circumstances the two-track approach leads to a set of well defined security requirements refined through the process implicit in the TCSEC, but which have little or no effect on the real system developed by the system developers. Systems developed using the two-track approach often encounter difficulties in obtaining the necessary security certifications and accreditations to process classified data.

Not only does the the two-track approach result in systems which cannot be accredited, but it is more costly, can result in less trust in a system composed of trusted and untrusted components, and inevitably results in tradeoffs between the two tracks as one track begins to dominate the other.

Early studies under this contract have concluded that it is essential to integrate security requirements and the software development process [4]. In addition, the Joint Logistics Command Orlando II conference panel VIII recommended that the DOD, "establish a committ^e to develop changes to DOD-STD-2167A that incorporate security requirements as an integral part of a system's development life cycle" [7]. To our knowledge such a panel has not been established. However, others in the security and software development communities are performing research aimed at integrating security requirements and DOD-STD-2167A. In order to advance these efforts an invitational workshop [8] was organized to provide a forum for key offices and personnel working in the areas of software development standards and information security. The goals of the workshop were to bring together technical researchers to share ideas on integrating software development standards, to identify the process and offices responsible for this, and finally to define future directions.

The results of the workshop indicate that there is a strong interest in defining approaches to including security requirements in the software development process. Discussion centered on how integration should occur. In particular, should DOD-STD-2167A be changed? Should security requirements be included as an appendix of DOD-STD-2167A? Should security requirements be included via a companion document such as DOD-STD-2168 [14]? Or should they be included through tailoring advice and perhaps new Data Item Description's (DID's)? There was not unanimous agreement on these approaches; however, it was pointed out that DOD-STD-2167A has been frozen for the next five years. Thus, revision is not a viable approach for the near-term.

2.1 2167A Explicit Development Process

DOD-STD-2167A describes a framework for a software development process that requires contractors to implement a process for managing the development of deliverable software. The major activities of the DOD-STD-2167A process include:

1. System Requirements Analysis/Design
2. Software Requirements Analysis
3. Preliminary Design
4. Detailed Design
5. Coding and Computer Software Unit (CSU) Testing
6. Computer Software Component (CSC) Integration and Testing
7. Computer Software Configuration Item (CSCI) Testing
8. System Integration and Testing

Within each of these activities DOD-STD-2167A further defines a set of reviews, a set of steps to be taken in support of the reviews, and a list of deliverables. These are summarized in Figures 1 and 2, taken directly from [13].

DOD-STD-2167A

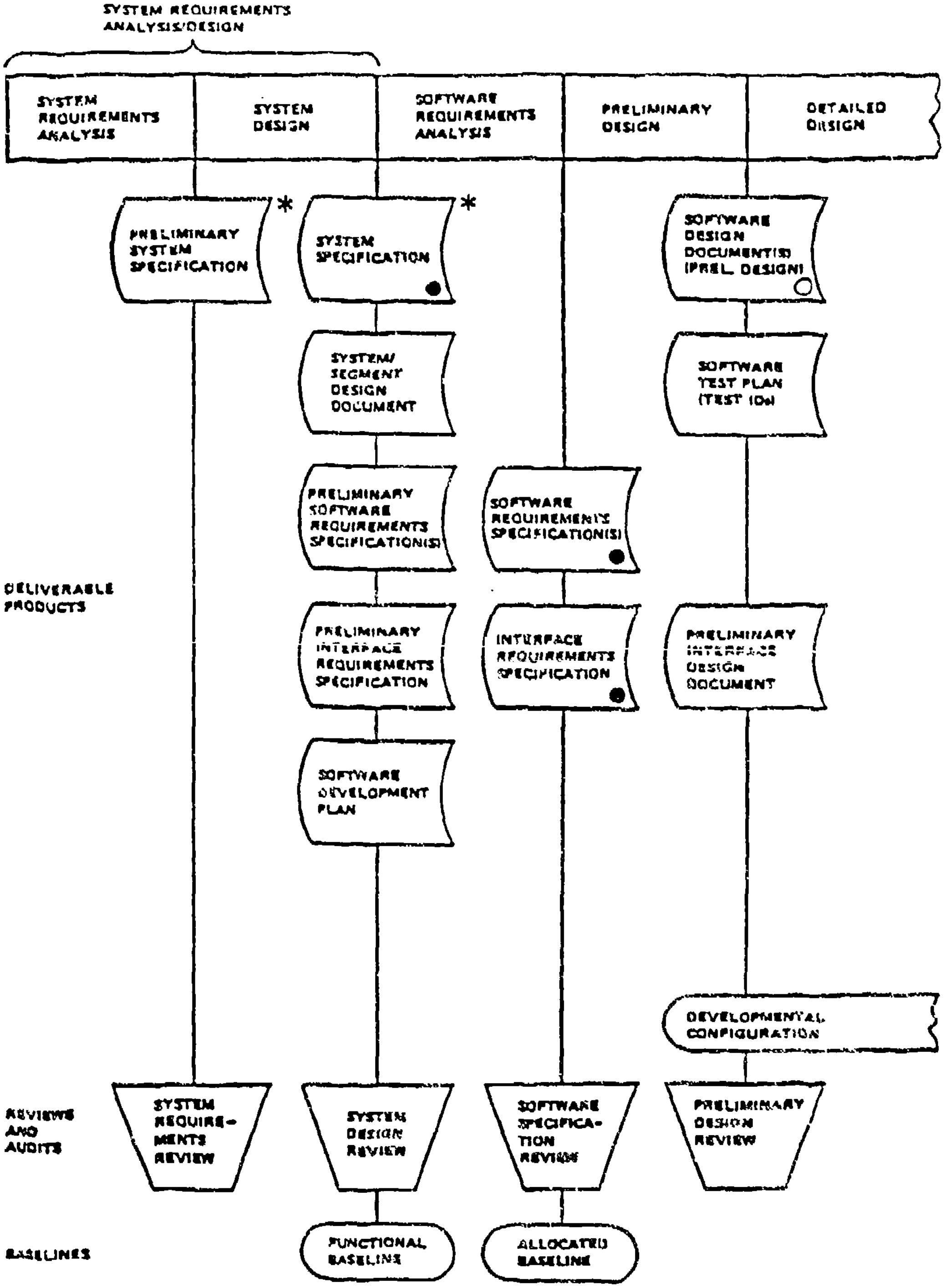

SYSTEM REQUIREMENTS ANALYSIS/DESIGN
SYSTEM REQUIREMENTS ANALYSIS
SYSTEM DESIGN
SOFTWARE REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
DETAILED DESIGN
PRELIMINARY SYSTEM SPECIFICATION *
SYSTEM SPECIFICATION ● *
SOFTWARE DESIGN DOCUMENT(S) (PREL. DESIGN) ○
SYSTEM/ SEGMENT DESIGN DOCUMENT
SOFTWARE TEST PLAN (TEST IDs)
PRELIMINARY SOFTWARE REQUIREMENTS SPECIFICATION(S)
SOFTWARE REQUIREMENTS SPECIFICATION(S) ●
DELIVERABLE PRODUCTS
PRELIMINARY INTERFACE REQUIREMENTS SPECIFICATION
INTERFACE REQUIREMENTS SPECIFICATION ●
PRELIMINARY INTERFACE DESIGN DOCUMENT
SOFTWARE DEVELOPMENT PLAN
DEVELOPMENTAL CONFIGURATION
REVIEWS AND AUDITS
SYSTEM REQUIRE-MENTS REVIEW
SYSTEM DESIGN REVIEW
SOFTWARE SPECIFICA-TION REVIEW
PRELIMINARY DESIGN REVIEW
BASELINES
FUNCTIONAL BASELINE
ALLOCATED BASELINE

Figure 1: Deliverable Products, Reviews, Audits, and Baselines

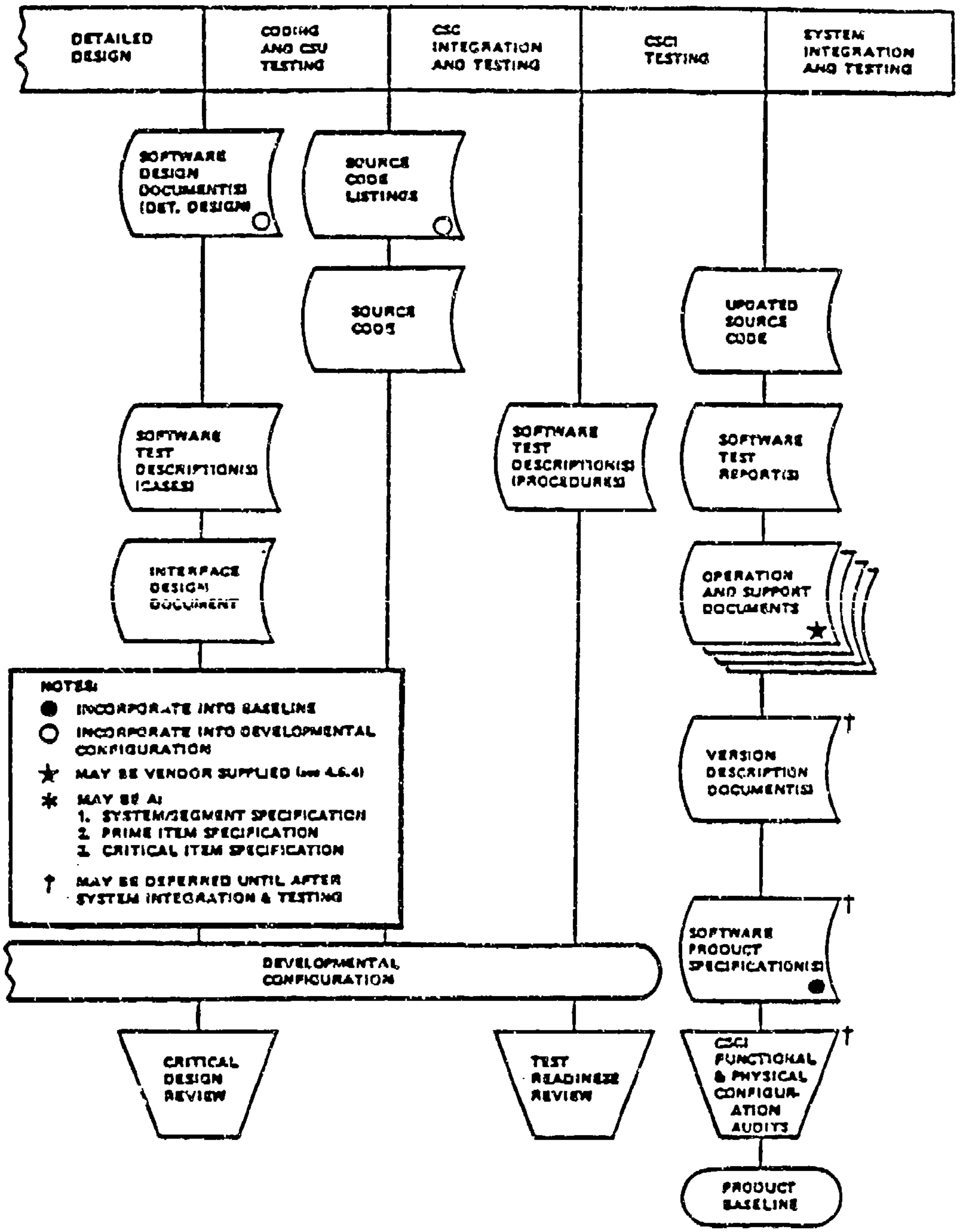

DETAILED DESIGN
CODING AND CSU TESTING
CSC INTEGRATION AND TESTING
CSCI TESTING
SYSTEM INTEGRATION AND TESTING
SOFTWARE DESIGN DOCUMENT(S) (DET. DESIGN)
SOURCE CODE LISTINGS
SOURCE CODE
UPDATED SOURCE CODE
SOFTWARE TEST DESCRIPTION(S) (CASES)
SOFTWARE TEST DESCRIPTION(S) (PROCEDURES)
SOFTWARE TEST REPORT(S)
INTERFACE DESIGN DOCUMENT
OPERATION AND SUPPORT DOCUMENTS
NOTES:
● INCORPORATE INTO BASELINE
○ INCORPORATE INTO DEVELOPMENTAL CONFIGURATION
★ MAY BE VENDOR SUPPLIED (see 4.6.4)
✱ MAY BE A:
1. SYSTEM/SEGMENT SPECIFICATION
2. PRIME ITEM SPECIFICATION
3. CRITICAL ITEM SPECIFICATION
† MAY BE DEFERRED UNTIL AFTER SYSTEM INTEGRATION & TESTING
VERSION DESCRIPTION DOCUMENT(S)
SOFTWARE PRODUCT SPECIFICATION(S)
DEVELOPMENTAL CONFIGURATION
CRITICAL DESIGN REVIEW
TEST READINESS REVIEW
CSCI FUNCTIONAL & PHYSICAL CONFIGURATION AUDITS
PRODUCT BASELINE

Figure 2: Deliverable Products, Reviews, Audits, and Baselines - continued

The System Requirements Analysis and Design Phase consists of describing a system design and system architecture. [1] This analysis is usually conducted on a Preliminary System Specification and is intended to determine whether the software requirements are consistent and complete. The design and system architecture are documented in a System/Segment Specification (SSS). In addition to the SSS, the contractor also develops a Software Development Plan (SDP) which states the contractor's plans for conducting the activities and producing the deliverables required by DOD-STD-2167A. This phase often occurs in conjunction with the Pre-System Development Phase, since a contractor's early design description is often required in a response to a Request for Proposals (RFP).

The Software Requirements Analysis Phase consists of defining the overall *software* architecture. This step allocates requirements to individual Computer Software Configuration Items, and documents the software architecture in the Software Requirements Specification (SRS). In addition to allocating requirements, it is necessary at this phase to define the complete set of interface requirements for the external interfaces.

The Preliminary Design Phase consists of defining a preliminary design for each Computer Software Configuration Item and further allocating requirements from the SRS and interface specification. This phase is intended to establish design requirements and to develop a preliminary design for the internal interfaces. These design decisions and any additional engineering information generated during the preliminary design process are documented in the Software Design Document (SDD).

The Detailed Design Phase consists of developing a detailed design by allocating requirements to individual Computer Software Units and establishing design requirements for each unit. In developing the design the contractor is expected to develop the detailed design of the external interfaces. The design decisions made during this phase are documented in the SDD. It should be noted that the SDD is a living document that exists across design phases. This is an important aspect of the DOD-STD-2167A development process, which encourages contractors to refine the design and report on it accurately, rather than writing "snap-shot" specifications as was the case under MIL-STD-490 [12]. In addition, during this phase the contractor establishes a Software Development File (SDF) for each Computer Software Unit. The Software Development File is used to record information related to the development or support of software. It usually includes design considerations and constraints, design documentation and data, schedule and status information, test requirements, test cases, test procedures, and test results. It is important to note that much of this material can be included in the SDF by reference to other documents.

[1] [2] notes that DOD-STD-2167A uses the term "system architecture" quite differently than does the TCSEC. In DOD-STD-2167A "system architecture" refers to the contractor's breakdown of the system into functional areas. In the TCSEC, "system architecture" refers to the requirement to structure the system and the TCB in order to increase assurance that the TCB satisfies the reference monitor requirements.

The Coding and Computer Software Unit Testing Phase consists of coding and testing each software unit. During this process the contractor must ensure that the algorithms and logic employed are correct and that the software satisfies its specified requirements. The test results are recorded in the Software Development Files (SDFs).

The Computer Software Component Integration and Testing Phase consists of testing and integration to ensure that the algorithms and logic employed are correct and that the integrated component satisfies its specified requirements. This is analogous to the previous phase but it involves integrating multiple software units into components and then testing the components. This phase often involves making changes to the design documentation and code which necessitate retesting and updating the Software Development Files (SDFs) of all software units and components. The procedures used for setting up, conducting, and analyzing the tests are documented in the Software Test Description (STD).

The Computer Software Configuration Item Testing Phase consists of formal qualification testing. The results of this testing are recorded in the Software Test Report (STR). Results of this testing often require revisions to the Software Design Documents (SDDs), code, and Software Development Files. Following successful completion of formal qualification testing the final source code is prepared for delivery as specified in the SRS. The delivery is accompanied by the final Software Product Specification (SPS) which is developed during this phase. In addition to preparing the source code and SPS for delivery, the software support and operational documentation is also prepared in preparation for transitioning the deliverable software from development to support. The software support and operational documentation consists of:

1. Computer Resources Integrated Support Document (CRISD)
2. Computer System Operator's Manual (CSOM)
3. Software User's Manual (SUM)
4. Software Programmer's Manual (SPM)
5. Firmware Support Manual (FSM)

The System Integration and Testing Phase involves supporting the Functional and Physical Configuration Audits and is the final step in the 2167A software development process.

2.2 TCSEC Development Paradigm

The TCSEC does not explicitly describe a framework for the software development process. Rather it embodies certain design principles implicitly. The TCSEC is intended as an evaluation criteria oriented towards the evaluability of a design, instead of the process used in the design. However, in order to achieve a design that can be evaluated at the B2 and higher levels of the TCSEC [2], it is necessary to follow an implicit design paradigm which consists of developing the following design documents and correspondences.

1. Philosophy of Protection
2. Security Policy Model
3. Formal Top-Level Specifications (FTLS)
4. Descriptive Top-Level Specifications (DTLS)
5. Security Policy Model to FTLS Correspondence
6. DTLS and FTLS Correspondence to Trusted Computing Base (TCB)
7. Covert Channel Analysis
8. Functional Testing
9. Security Testing
10. Security Specific Documentation
    - (a) Trusted Facility Manual
    - (b) Security Features User's Guide
    - (c) Configuration Management Plan

Each of the above documents and correspondences is intended to ensure that the proper security requirements are addressed in the design of a Trusted Computing Base (TCB). While the TCSEC does not require that these are produced in the order listed above, it is

[2] The TCSEC is divided into four divisions: D, C, B, and A ordered in a hierarchical manner with the highest division (A) being reserved for systems providing the most comprehensive security. Each division represents increased confidence in the system for protection of sensitive information. The discussion of TCSEC requirements in this section focuses primarily on the TCSEC requirements for B2-A1 systems since for integrity reasons it is believed that complex battle management systems will require at least B2 systems.

far more difficult to fully satisfy the TCSEC assurance requirements if they are not produced in roughly this order.

The Philosophy of Protection is intended to capture the essential security requirements of the system (e.g. access control) and how they are translated into the TCB. This is an informal document which is used to identify the specific TCB protection mechanisms.

Once the essential security requirements and corresponding protection mechanisms have been identified, a formal model of the security policy can be developed. The formal model is a mathematically precise statement of the security policy for the system under development. Formal models, such as the well known Bell and La Padula Model [1], are often stated in terms of an abstract model and a concrete model [11]. The abstract model captures the essential security requirements, (e.g. *-property, Simple Security) while the concrete model provides an abstract set of rules of operation (e.g. Get Read Access).

The abstract rules of operation can then be elaborated into a high-level design specification in the form of a Descriptive Top-Level Specification (DTLS). The TCSEC defines a Top-Level Specification as "a non-procedural description of system behavior at the most abstract level; typically, a functional specification that omits all implementation details." A DTLS is written in an informal language (e.g. English), a program design language, or a combination of the two. The DTLS must completely and accurately specify the TCB interface in terms of exceptions, error messages, and effects. The DTLS is intended to capture the user-visible actions of the TCB. One common approach to developing a DTLS is to write informal descriptions of the TCB functions in terms of input, processing, and output statements.

The highest level of assurance in the TCSEC, A1, requires that a Formal Top-Level Specification (FTLS) be developed. The FTLS is written in a formal specification language and must be proven to enforce the security policy as described by the formal model. Because most common formal specification languages can not be used to specify temporal properties and subtle hardware characteristics, the FTLS is not required to provide a *complete* description of the TCB interface. Instead the FTLS must only provide an *accurate* description of the TCB interface. Thus, it is important to note that one needs to refer to *both* the FTLS and the DTLS in order to fully specify the system under development.

To gain assurance that the system design will enforce the Security Policy, the FTLS is shown, through a combination of formal and informal techniques, to be consistent with the formal model. This consistency proof is often referred to as the FTLS to Model Correspondence.

Once the design has been shown to be consistent with the security policy (via the FTLS-Model Correspondence), it is necessary to establish that the implemented system (the TCB) is consistent with the design. This is done informally and requires establishing the correspondence between both the DTLS *and* the FTLS, and the TCB. It is necessary to use both the DTLS and the FTLS, since the FTLS provides better assurance through its formalisms, although as noted above it is not complete.

Additional assurance of the system's security is gained through a Covert Channel Analysis. A covert channel is defined by the TCSEC to be "any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy" [5]. For TCSEC B-Level systems the covert channel analysis is informal and is performed on the design documents and system implementation. At the A1-Level of the TCSEC formal techniques are used and the covert channel analysis is usually performed on the FTLS.

In the TCSEC development paradigm there are two types of testing requirements: Functional Testing, and Security Testing. Functional Testing is similar to that required by DOD-STD-2167A and is aimed at demonstrating that the system meets its specifications. Security Testing, sometimes called Penetration Testing, is intended to show that not only does the system do what it is intended to, but that it does nothing else. In particular, Security Testing attempts to, "uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB" [5].

In addition to the design documents and correspondences described above, the TCSEC requires several documents that are security specific. These are:

- A Trusted Facility Manual addressed to the ADP system administrator and which presents cautions about functions and privileges that should be controlled when running a secure facility.
- A Security Features User's Guide which describes the protection mechanisms provided by the TCB and presents guidelines on their use.
- A Configuration Management Plan which describes the configuration management procedures used for controlling changes to the system during its entire life-cycle.

2.3 Relationship between 2167A and TCSEC

An initial examination of these two sets of requirements (DOD-STD-2167A and the TCSEC) might lead one to believe that there is little relationship between the two processes, and thus any software development that must satisfy both sets of requirements could well proceed along the two paths independently. As discussed earlier this is in fact the approach that most efforts have taken in recent years. However, this approach led to many problems. The primary problem has been difficulty in obtaining the necessary security certifications and accreditations because the implemented system did not correspond to the security assurance evidence or the security assurance evidence was lacking or insufficient. It is believed that in order for large complex battle management systems to be developed using the DOD-STD-2167A software development process *and* meet security requirements an integrated approach must be developed.

Closer examination of the two processes shows that in fact there are many parallels. Both rely on hierarchical decomposition, refinement of requirements into implementation (SSS-SRS-SDD and Model-FTLS-DTLS), and testing and specification correspondences play a key role in both processes. Given these similarities and the driving need to develop trusted systems using the DOD-STD-2167A software development process, an integrated approach is not only feasible but is highly desirable.

The next section will propose a first approximation at a tailored software development process which integrates security requirements and DOD-STD-2167A requirements.

3 Integration Issues

There are several issues surrounding software development models which need to be addressed before an integrated trusted software development process can be proposed. It is important to recognize that while DOD-STD-2167A describes an explicit software development*process*, it does not prescribe a specific underlying software development *model*. This has been a point of some confusion and controversy. The software development process of DOD-STD-2167A describes a set of phases for software development and a set of deliverables and reviews relative to the phases. The choice of a particular software development model (e.g. waterfall or spiral) is left up to the contractor [3]. A software development model provides a framework for guiding the software development process. The two most common and often debated software development models are the waterfall model [9] and the spiral model [3]. The waterfall model treats the software development process as a series of sequential steps each of which is completed before the next step is begun. The spiral model is a risk-driven approach which focuses on identification and reduction of risk during the development process by iterating over phases of the software development process.

[3] This is noted in the foreword to the standard which says, " This standard is not intended to specify or discourage the use of any particular software development method. The contractor is responsible for selecting software development methods (for example rapid prototyping) that best support the achievement of contract requirements."

DOD-STD-2167A implicitly imposes a hierarchical decomposition structure on the software development process, as depicted in Figure 3 from [13]. It is this imposition of hierarchical decomposition that leads many to believe that DOD-STD-2167A must be used in accordance with a traditional waterfall software development model, and that the standard is not suitable for use in a software development effort where a prototyping or spiral model of software development is employed [10]. However, it should be noted that the DOD-STD-2167A process has numerous revisions and iterations built into it, for example the requirements of the CSU Testing phase; thus, the dichotomy between the spiral development model and DOD-STD-2167A is not as great as it may first appear. As it turns out, iteration within the software development process of DOD-STD-2167A is a crucial point with respect to defining an integrated approach.

Secondly, it is important to recognize that security requirements affect *all* stages of software development. Security analysis is by its very nature iterative because it consists of examining an evolving system at various stages in order to detect security weaknesses which can then be removed from the system design. Once the identified security weaknesses are repaired then the design is once again analyzed, and design proceeds in this manner. Because the TCSEC requires that a direct correspondence be shown between the code and the top-level design specifications (FTLS and DTLS) this iterative process continues throughout all stages until the final system implementation is completed.

Therefore, the approach to integrating security requirements and DOD-STD-2167A presented in the next section assumes that it is both feasible and desirable to use DOD-STD-2167A in an iterative manner.

4 Integration Approach

The integration approach presented in this section assumes that both the process of developing software and the engineering of secure systems are iterative processes. However, in order to facilitate intermediate deliveries of system design and documentation it is desirable to confine the iteration to within specific intervals. Thus several cut points have been identified which can isolate the affects of the iteration to the components of a specific interval. It should be noted that these intervals do not necessarily correspond to the software development phases and reviews of DOD-STD-2167A. In adopting such an iterative approach it is necessary to recognize that final versions of certain system specifications may be delivered late in the life cycle of the system development.

This is a first approximation of where, when, and how the software development process of

DOD-STD-2167A

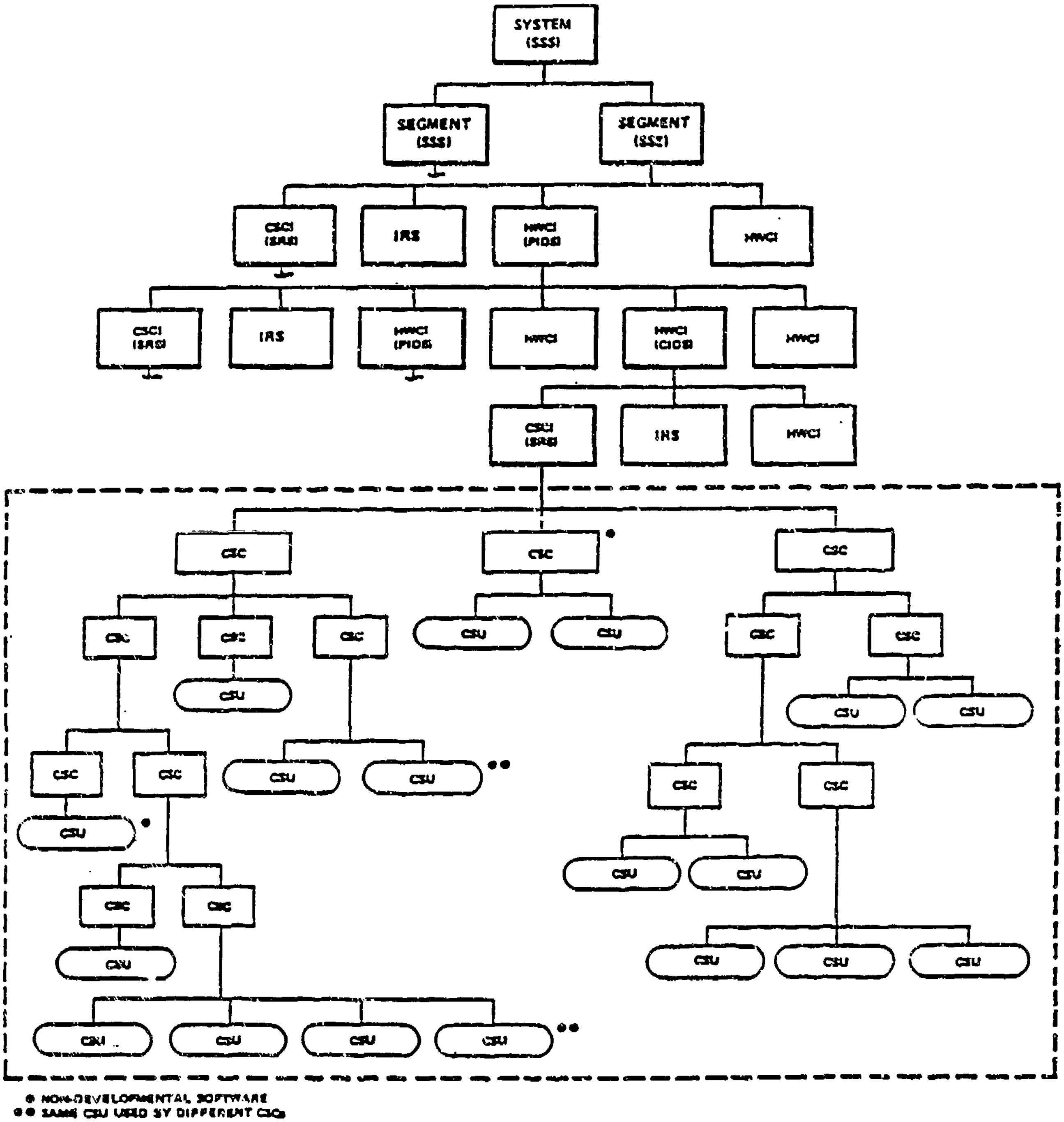


Figure 3: Example of a System Breakdown and CSCI Decomposition

DOD-STD-2167A and security requirements should be integrated; further discussion, examination, and experimentation are anticipated. For purposes of initial analysis, we used the security requirements of the TCSEC, since the confidentiality requirements of the TCSEC are better understood than the emerging security requirements pertaining to integrity and assured service. It is believed that this approach is general enough so that it can be extended to include the additional security requirements of integrity and assured service as they become better defined. For example, identification of where in the process a formal (confidentiality) model should occur applies equally well to a formal model of integrity.

In defining an approach to integrating the two processes, three different aspects were examined. First, one needed to consider the two timelines and determine when in the integrated development process various phases should occur. Secondly, it was necessary to examine the specific requirements, documents and deliverables in order to determine what new requirements were introduced as a result of integrating the processes and which existing requirements needed to be tailored. Finally, since the integration relies on an iterative model of software development it was necessary to determine the intervals which involved iteration.

4.1 The Timelines

The software development timeline of DOD-STD-2167A was used as a basis for forming an integrated software development process, because it explicitly identifies phases. The design phases, documentation, and correspondences of the TCSEC were then mapped into the phases and deliverables required by DOD-STD-2167A. The resulting integrated trusted software development approach is shown in Figure 4. Not all TCSEC processes could be directly mapped into existing 2167A deliverables and this is depicted by the shaded boxes and broken lines in the figure.

The remainder of this section will describe each of the TCSEC requirements and the rationale behind its placement in a phase of the DOD-STD-2167A timeline.

The Philosophy of Protection should be developed during the Systems Requirements Analysis Phase along with the development of the SSS. This is because, as noted previously, the Philosophy of Protection is intended to capture the essential security requirements. Thus, prior to entering the design process it is necessary to identify and document the overall *system* security requirements. This is consistent with the type of activities required by DOD-STD-2167A during the System Requirements Analysis Phase, which consists of describing an overall system design and architecture.

The formal model should be developed during the Software Requirements Analysis Phase

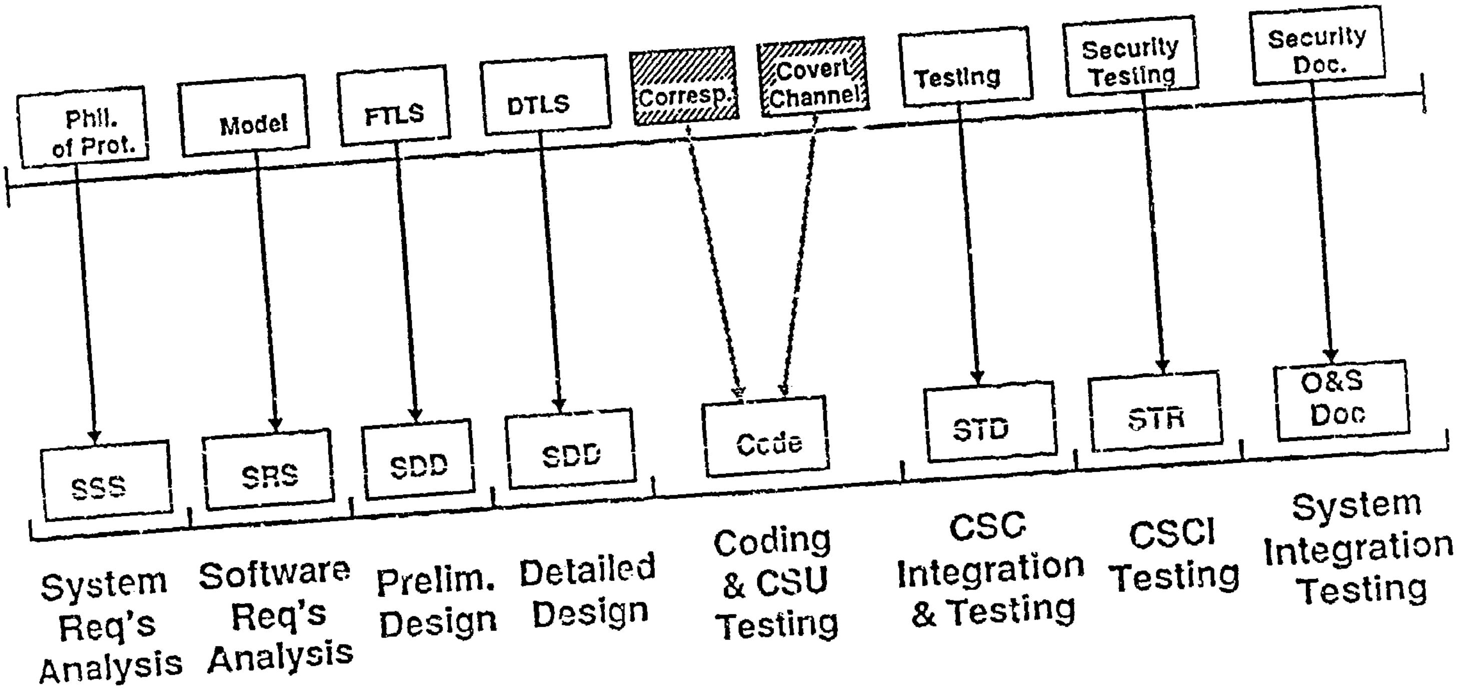


Figure 4: TCSEC and DOD-STD-2167A Software Development Timelines

along with the development of the SRS. This is because the Software Requirements Phase focuses on establishing software design requirements and constraints (through the interface requirements specification), and because the purpose of a formal model is to establish abstract design requirements and constraints. It is important that the formal model be written early in the development cycle in order to demonstrate that developers have a clear understanding of the security requirements and that the security requirements are sound and consistent. The formal model of the security requirements can then drive the development of the specifications.

The FTLS is developed during the Preliminary Design Phase in conjunction with the SDD. This phase focuses on establishing design requirements on a component level, which is similar to the goal of an FTLS to provide an abstract design of the functions of the TCB. It should be possible to perform the initial FTLS-Model correspondence during this phase. However, the development of the DTLS and completion of the SDD may require revisions to the FTLS and reconstruction of the FTLS-Model correspondence.

The DTLS is developed during the Detailed Design Phase in conjunction with the SDD. Both the DTLS and the SDD document design decisions pertaining to function interfaces, exceptions, error messages, and effects. The similarity between the information in the DTLS of a B2-A1 level TCB and the information in an SDD for a highly trusted system make the Detailed Design Phase a high leverage point for integrating security requirements into the software development process. In fact, the one point at which systems developed under the two-track approach tend to overlap is in the description of the design in the SDD and the DTLS. Tremendous savings and increased assurance can be gained by integrating these design decisions and documents.

It is an interesting observation that there is no TCSEC design process which directly maps to the DOD-STD-2167A Coding and CSU Testing Phase. This is largely due to the TCSEC's emphasis on design rather than implementation. The TCSEC does however require that the FTLS and DTLS be shown to correspond to the code. This correspondence and the Covert Channel Analysis should occur during the Coding and CSU Testing Phase which is concerned with testing components for correspondence to their specifications. This phase in both the TCSEC development paradigm and the DOD-STD-2167A development process has the greatest amount of iteration involved, and revisions required at this stage could conceivably affect all previous stages. This will be discussed further below.

The TCSEC functional testing can map directly into the DOD-STD-2167A CSC Integration and Testing phase since both are concerned with testing that the system works as claimed and that it meets its specifications.

Security Testing cannot begin until the CSCI Testing Phase and cannot be completed until the Final System and Integration Testing Phase. This is because the types of subtle flaws that Security Testing aims to discover may not be present until all CSCIs in the system are integrated.

Finally, the Security Specific Documentation should be delivered during the System Integration and Testing Phase along with the operation and support documentation. Earlier drafts of these documents can be developed during the design phases. However, they are subject to revision as a result of the testing and correspondences.

The above discussion has demonstrated that there is a natural integration of the TCSEC security requirements and the DOD-STD-2167A software development process. In most cases this integration requires tailoring or modifying the DOD-STD-2167A deliverables. In some cases new security requirements and deliverables are introduced. The next subsection will discuss tailored and new deliverables.

4.2 Tailored and New Requirements

One useful outcome of the invitational workshop mentioned in section 2 was a discussion of the trade-offs involved in modifying versus creating new deliverables. In defining an integrated approach, it at first appeared that one should create new deliverables for all of the security specific evidence. However, discussion at the workshop indicated that there could be a significant risk to creating new deliverables. In particular, if the cost (in terms of level of effort or dollars) is negotiated between the government and the contractor then it is quite likely that new or additional deliverables above those specificed in DOD-STD-2167A might be eliminated to save time or money. Thus, it was concluded that wherever possible existing DOD-STD-2167A deliverables should be tailored instead of creating new deliverables. There are several advantages to this approach. First, DOD-STD-2167A is already structured to provide for tailoring of deliverables to specific contracts. Second, this approach minimizes the likelihood of security deliverables being cut. Third, the tailoring of deliverables leads to a better integrated software development process which minimizes the chances of a two-track approach being used. Close examination of the integrated approach presented in this paper confirmed that the DOD-STD-2167A deliverables shown in Table 1 can be tailored to incorporate TCSEC requirements.

However, several security requirements from the TCSEC were not easily incorporated into the existing DOD-STD-2167A deliverables and are shown in Table 2. Furthermore, it is anticipated that as the integrity and assured service requirements of security become better defined new deliverables will arise. Whether or not the new deliverables can be incorporated into existing 2167A deliverables will have to be determined as the deliverables arise.

Tailored 2167A Deliverable	Incorporated TCSEC Requirement
SSS	Philosophy of Protection
SRS	Security Policy Model
SDD	DTLS and FTLS
SDF	Functional Testing
STR	Security Testing
CSOM	Trusted Facility Manual
SUM	Security Features User's Guide
SDP	Configuration Management Plan

Table 1: Tailored Deliverables

New Security Deliverables
Model-FTLS Correspondence
FTLS-Code Correspondence
DTLS-TCB Correspondence
Covert Channel Analysis

Table 2: Security-Specific Deliverables

4.3 Where Does Iteration Occur

Section 3 discussed why it was desirable to define the integrated software development approach with respect to an iterative software development model. While it is believed that iteration and revision can lead to a better specified and implemented system, we also recognize that government acquistion authorities and certification/accreditation agencies need intermediate deliverables in order to assess a system under development.

We have therefore attempted to identify points at which certain deliverables can be frozen and delivered, thereby isolating the revision and iteration process to specific intervals. Initial analysis of the integrated software development approach identified three major intervals within which iteration can be contained. These are shown in Figure 5 and discussed below.

The first interval consists of two DOD-2167A phases: System Requirements Analysis/Design, and Software Requirements Analysis. These two phases are focused on defining overall system and software requirements and identifying the essential security requirements and constraints. During these two phases, the SSS (tailored to include the Philosophy of Protection) and the SRS (tailored to include the formal model) are developed. These documents may require several versions as better understanding of the system and security requirements

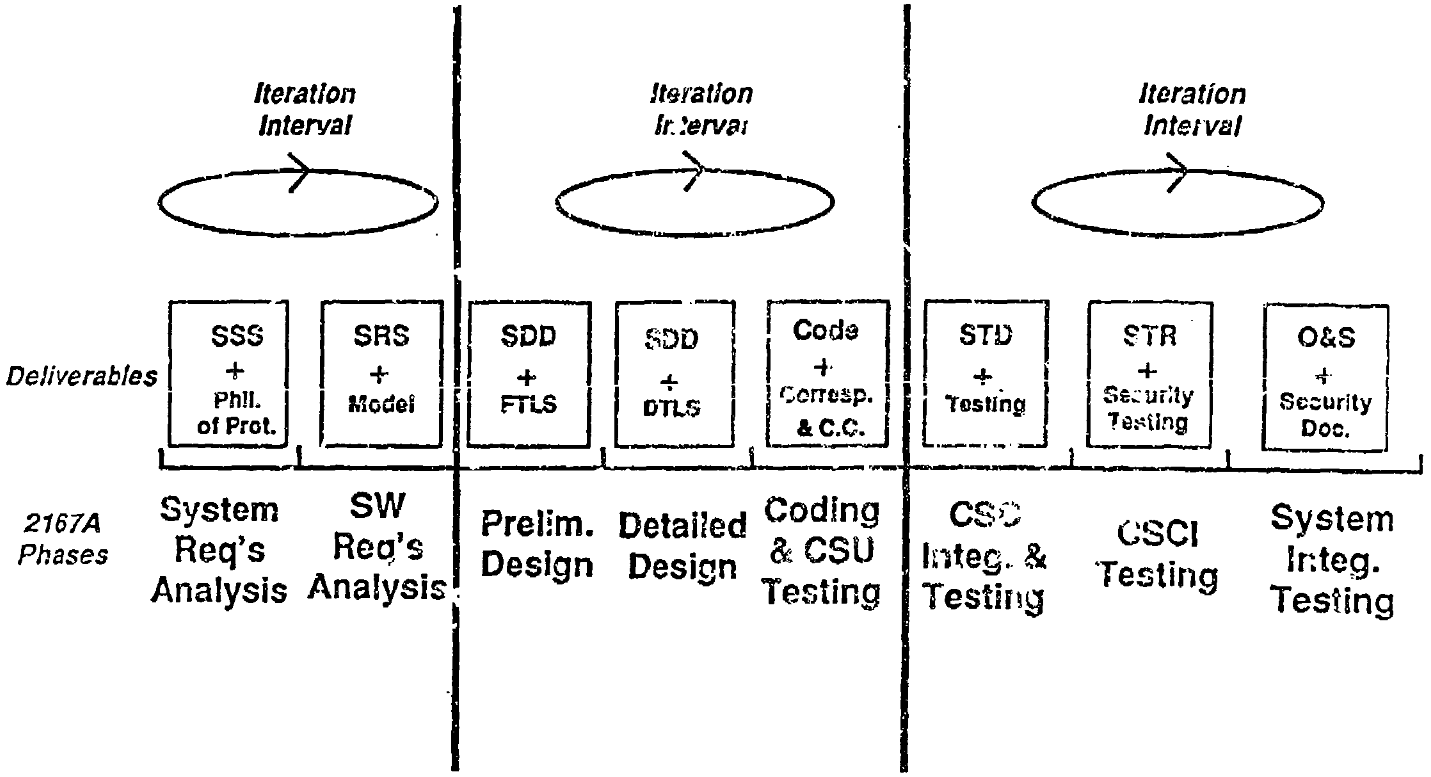


Figure 5: Iteration Intervals within the Integrated Approach

emerges when the Formal Model and SRS are developed. In order to ensure that all parties have a common understanding of the system to be developed, it is desirable to establish baseline versions of the SSS and SRS. These documents should be complete and a first draft incorporated into the baseline at the Software Specification Review. Ideally, the formal model could be frozen during this interval and used to drive the specification process.

However, this is not entirely realistic since the complex systems under development today have many more capabilities and design constraints than only those captured by the formal model. Therefore, it is recommended that the most abstract portion of the model be completed during this interval, but that the concrete portion of the model be revised during the next interval as the complete system design is specified.

The second interval consists of three DOD-STD-2167A design phases: Preliminary Design, Detailed Design, and Coding and CSU Testing. These phases are focused on establishing, refining and implementing the system design. During this interval the SDD (tailored to include the DTLS and FTLS), the Code and the Correspondences are developed. As noted previously the results of the correspondences, covert channel analysis, coding, and CSU Testing will probably require revisions to the Code and SDD. These documents should be completed and delivered in draft form at the end of the Coding and CSU Testing phase. However, they cannot be finalized until the System Integration and Testing Phase due to the possibility of changes required as a result of the Functional and Security testing.

The third and final interval consists of three DOD-STD-2167A design phases: CSC Integration and Testing, CSCI Testing, and System Integration and Testing. These phases are focused on demonstrating that the implemented system meets its specifications, and on developing operation and support documentation. During this interval the system and all system specifications and design documents are finalized and delivered.

5 Conclusions

This paper has examined the software development processes of DOD-STD-2167A and the TCSEC. It was determined that in order to integrate these two processes it is necessary to view both the process of developing software and the engineering of secure systems as iterative processes.

A first approximation at a tailored software development process which integrates security requirements and DOD-STD-2167A requirements was presented, along with the rationale for the integration. This demonstrated that there is a natural integration. Furthermore, the integrated approach appears to be practical and realizable in the near-term since it relies on tailoring DOD-STD-2167A, rather than requiring major revisions to the standard. For the immediate future, this approach should be subjected to peer review and used on a development project which requires conformance to the DOD-STD-2167A requirements and TCSEC security requirements. It is becoming increasingly clear that in order for large complex battle management systems to be developed using the DOD-STD-2167A software development process *and* meet security requirements, an integrated development approach such as the one suggested here must be followed.

A List of Acronyms and Abbreviations

ADP Automatic Data Processing

CRISD Computer Resources Integrated Support Document

CSC Computer Software Component

CSCI Computer Software Configuration Item

CSOM Computer Software Operator's Manual

CSU Computer Software Unit

DID Data Item Description

DTLS Descriptive Top-Level Specification

FSM Firmware Support Manual

FTLS Formal Top-Level Specification

RFP Request For Proposal

SDD Software Design Document

SDF Software Development File

SDP Software Development Plan

SPM Software Programmer's Manual

SPS Software Product Specification

SRS Software Requirements Specification

SSS System/Segment Specification

STD Software Test Description

STR Software Test Report

SUM Software User's Manual

TCB Trusted Computing Base

TCSEC Trusted Computer Systems Evaluation Criteria

References

[1] Bell, D. E. and LaPadula, L.J., *"Secure Computer Systems: Unified Exposition and Multics Interpretation"*, MTR-2997, The MITRE Corp, Bedford, MA, March 1976.

[2] Bodeau, D.J., *"TCSEC Specification and Verification Documentation Applicability: Interim Report"*, WP-27545, The MITRE Corporation, Bedford, MA, September 1987.

[3] Boehm, B.W., *"A Spiral Model of Software Development and Enhancement"*, reprinted in Software Engineering Notes, Association for Computing Machinery, Volume 11, No 4, August 1986.

[4] Crocker, S.D., Siarkiewicz, E., *"Software Methodology For Development of a Trusted BMS: Identification of Critical Problems"* TM-8361/003/00, Camarillo, CA, The UNISYS Corporation, April 1988.

[5] *"Department of Defense Trusted Computer System Evaluation Criteria,"* DOD 5200.28-STD, December 1985.

[6] Farmer, W.M., D.M. Johnson, F.J. Thayer, *"Review of RAP Design Verification"*, MTR-10227, The MITRE Corporation, Bedford, MA, April 1987.

[7] *"Proceedings JLC-CRM 4th Biennial Software Workshop, Orlando II"*, 23 March 1987.

[8] *"Minutes of the Trusted Software Development Workshop 21 October 1988"*, Pfleeger, C.P., T.C.V. Benzel, L.D. Martin, TIS-R-197, Trusted Inf rmation Systems Inc., Glenwood, MD, 14 December 1988.

[9] Royce, W.W., *"Managing the Development of Large Software Systems: Concepts and Techniques"*, Proceedings, WESCON, August 1970.

[10] Marmor-Squires, A.B., Rougeau, A.P., *"Issues in Process Models and Integrated Environment for Trusted System Development"*, Proceedings 11th National Computer Security Conference, Baltimore, MD, October 1988.

[11] Tavilla, D.A., *"A Guide to Understanding the Orange Book Security Model Requirements"*, WP-26782, The MITRE Corporation, Bedford, MA, May 1986.

[12] *MIL-STD-490A, Military Standard: Specification Practices*, Department of Defense, 4 June 1985.

[13] *DOD-STD-2167A, Military Standard: Defense System Sofiware Development,* 29 February 1988.

[14] *DOD-STD-2168, Military Standard: Defense System Software Quality Program*, 29 April 1988.

THE ELECTRONIC SECURITY COMMAND AUTOMATED ACCREDITATION PACKAGE

MR HORACE B. PEELE
Chief, Policy and Security
ESC/Communications-Computer Systems
San Antonio, Texas 78243-5000

INTRODUCTION

The Electronic Security Command (ESC) is one of thirteen major commands within the United States Air Force. It performs several classified intelligence missions. As the ESC Designated Approving Authority (DAA) for sensitive unclassified and collateral systems and as the official liaison office to a national-level agency for operational computer security issues, I am pleased to have this opportunity to discuss an important ESC initiative, the development of the "ESC Accreditation Package".

BACKGROUND

First, we must set the stage. As the need for computers grew in numbers and the interdependency between information processing and telecommunications increased, the Air Force realized that several potential new threats were developing:

First, the end-users were becoming more and more reliant on automated systems to support critical missions,

Second, there is an increased exposure to risk due to requirements for networking to support these missions,

And lastly, the structuring of communications-computer systems with well defined data bases results in a high loss potential if these systems are exploited.

In other areas of the federal government and the commercial world, the same type of concerns began to grow. In an effort to strengthen its resources against these weaknesses, the Air Force functionally relocated and redefined "COMPUSEC" to be part of Information Systems Security defined as: "The protection afforded to communications and computer systems in order to preserve the availability, integrity, and confidentiality of the systems and the information contained within the system. Such protection is the application of COMSEC, TEMPEST, and computer security executed in liaison with information security, personnel security, industrial security, resources protection, and physical security."

This was the first effort within the Air Force to integrate COMSEC, TEMPEST and COMPUSEC. Later, the Air Force changed the title "Information Systems Security" (ISS) to "Communications-Computer Systems Security" without further redefinition. Partial rationale was to deconflict and distinguish ISS and the already existing

"Information Security", one of the fundamental security disciplines.

Information Security is defined as: The result of any system of administrative policies and procedures for identifying, controlling, and protecting from unauthorized disclosure, information whose protection is authorized by executive order or statue (DOD 5200.1-R/AFR 205-1).

FUNDAMENTAL SECURITY DISCIPLINES

For a complete understanding of ISS, it is necessary to place the fundamental security disciplines into context and describe their ISS relationships. One must recognize and be assured that there are only three fundamental security disciplines--"Personnel Security, Physical Security, and Information Security" and that all other security disciplines, without exception, are derived from and directly support one or more of these basic fundamental securities. If all written word were still chiseled into stone, the fundamental securities would be all that prevailed. It is only through technology that we transgress into the concept of derived security disciplines. Let's visit these fundamental securities by definition.

Personnel Security is a fundamental **single-disciplinary** security umbrella governing the establishment of policies and procedures to ensure that the acceptance and retention of employees (both military and civilian) and that granting access to classified information to those employees are clearly consistent with the interests of national security.

Physical Security is a fundamental **single-disciplinary** security umbrella governing the establishment of policies and procedures for an area that deals in terms of threats of physical damage to Air Force priority resources, safeguarding defense information, security against esponiage and subversion, and the USAF Resources Protection Program.

Information Security is a fundamental **single-disciplinary** security umbrella governing the establishment of policy relating to the protection of information, regardless of its physical state, which includes policy for unclassified, sensitive-unclassified, and classified information.

As well as understanding the fundamental securities, it is also necessary to relate how ISS, a derived security, supports the basic fundamental securities.

Information Systems Security is a derived **multidisciplinary** security umbrella governing the establishment of policy relating to the protection of information while the information is specifically in the electromagnetic state. It is a function which integrates Communications Security, Computer Security, and TEMPEST in direct interdisciplinary support of Personnel Security and Physical Security as well as Information Security.

Now that we have explored one way of looking at ISS, we need to discuss another issue which consistently interrupts the logic process, that being the misuse of certain words relating to ISS. Specifically, there is gross misuse and interchange of the words "certification" and "accreditation". This has been a topic of discussion during recent meetings of various Subcommittee on Automated Information Systems Security (SAISS) working groups.

The Chairman of the SAISS Policy Working Group indicated that while many departments and agencies do have policies and techniques for certification and accreditation, the problem is inconsistency. What one agency calls certification, another calls accreditation. The Chairman of the SIASS Systems Security Standards Working Group pointed out that there is no national-level policy outside the intelligence community requiring both certification and accreditation. It has been suggested that a new Executive Order be written to cover this issue.

In addressing any forum, one must be careful not to use these words incorrectly and to challenge the audience in order to place the briefing or discussion on common grounds. Therefore, let us visit these critical words and view their Air Force definitions.

Certification. A statement, based on detailed technical analysis, that specifies the extent to which a system meets security requirements (AFR 700-10, par. A1-5). The term is usually used in a phrase such as "TEMPEST certified" or "certified Trusted Computing System".

Accreditation. The official authorization granted by the appropriate Designated Approval Authority (DAA) permitting the processing of classified data on a communications-computer system (AFR 700-10, paragraph A1-4). The issuance of any approval is based upon the DAA's review of the system accreditation package.

From these points of view, "certification" is the proof that the system including all hardware and software actually works! While "Accreditation" is a "Mother, may I use it?" concept. This sets the stage for our discussion.

ACCREDITATION DISCUSSION

What format is an accreditation package? What life cycle phase does one submit the accreditation request? How does one obtain permission to use a system? Who is the Designated Approving Authority? How long does it take to get approval? Who could use an automated accreditation system? Many questions...

The proliferation of standalone systems, coupled with the accreditation requirement, has resulted in the production of literally hundreds of accreditation packages--for new systems, for systems which have had hardware or software configuration changes, and for systems which have been relocated from one place to another. These packages are all being created or updated, as the case may be, by the Information Systems Security Officer (ISSO) in some handy non-standard format and probably as an additional duty.

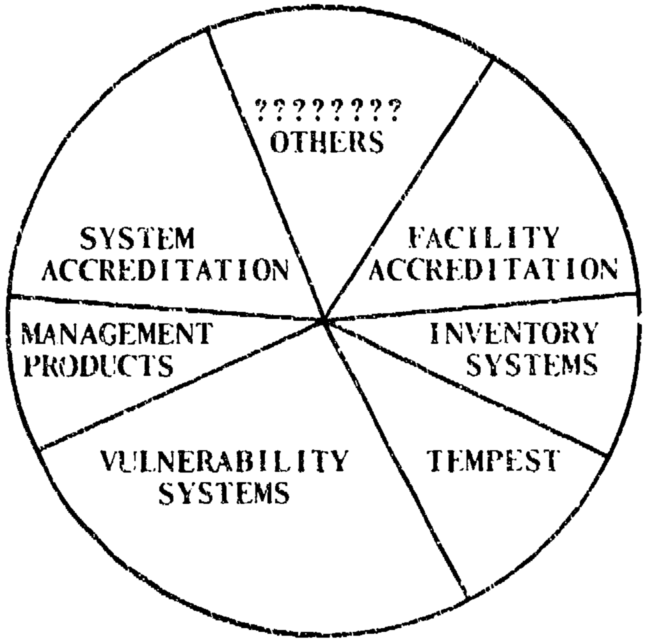
As workers, we understand the problem of being told to do something extra, the "NOT IN MY JOB DESCRIPTION" syndrone, and how it tends to deflate morale. As managers, we sometimes have no choice, especially when we are extremely short of full-time ISS personnel and have to do the manhour-intensive job out-of-hide.

Accreditation results in granting the user the approval-to-process. As a result, accreditation has received constant high-level attention reflecting the growing concern of managing increased security requirements with a continuing critical shortage of manpower. The fact that there is no standard format or process is also a major concern.

After the acquisition, the certification testing, and the permanent installation of the AIS, the accreditation process is the most critical to a successful operation. If the formal submission of the accreditation package is rejected by the DAA because it was improperly accomplished, then it usually results in a delay in implementing the new system, potentially a mission critical system, possibly even a life-saving system.

Upon preliminary investigation, it appeared that standardization could support interfaces to other systems such as: Vulnerability analysis; incident reporting analysis; national and lower-level data call responses; the creation of security management products; configuration control; inventory control; ease the requirements in host-tenant agreements; Public Law 100-235 reporting; and, other reporting requirements within the intelligence community. Simply stated, the requirements analysis revealed that accreditation is a piece of a large complex system of disjointed processes.

PIECES OF THE PIE



Automating the procedure could also reduce user complaints about workload, amount of time. But more beneficially, it would tend to capture the original input in electronic media form and assist in eliminating redundancy in preparation of reports among similar systems. Therefore, developing a standard accreditation package format and a user-interactive application operating on standard hardware systems was a giant step toward alleviating the burden of the ISSO and the system users.

The strategy for developing such a tool must be approached with caution. A failure could result if the development of a system does not consider all potential users. In the case of ESC, it was necessary to consider support to the accreditors from the both the classified and unclassified worlds.

The resultant system also had to support all organizational levels from the lowest-level unit up through to the major command level, to the military department, and to the national level agencies. The ESC system was designed and programmed using dBase III Plus operating under MS-DOS. It was designed through several DOD-wide workshops of drafting the knowledge of many representatives of many components. The purpose of the DOD-wide workshops was to standardize the format such that it would work for any Army, Navy or Air Force DAA. Next, the first dBase III Plus prototype was developed and sent to all Air Force major commands for evaluation. As a result, the prototype was expanded during development to meet all known accreditor requirements. More recently, it has been updated to support the PL 100-235 and new Director of Central Intelligence reporting requirements.

With a prototype program in hand, it was decided to determine the actual cost savings in the preparation of accreditation packages. A subordinate ESC organization was chosen as the test facility. This organization, comprised of approximately 450 personnel, had completed 85 accreditation packages on systems ranging from standalone personal computers to mainframe systems in one year. The average time in the preparation of these packages was 8 hours each. The following depicts the cost of their preparation.

8 Hours X 85 packages (Total hours)	680
The ISSO salary per hour	$22.00
Total cost of manual preparation	$14,960.00

The following evaluation was performed to determine the dollar value savings if the same 85 packages had been prepared using the dBase III Plus prototype system averaging 2 hours for each package.

2 hours X 85 packages (Total hours)	170
The ISSO salary per hour	$22.00
Total cost of dBase III prepara.ion	$3,740.00

A comparison reveals an outstanding result!

Total cost of manual preparation	$14,960.00
Total cost of dBase III preparation	3,740.00
Savings (one Year - one organization)	$11,220.00

One has to ask the following question: IF THIS BE TRUE, THEN WHAT IS THE DOLLAR VALUE TO THE AIR FORCE? WHAT IS THE DOLLAR VALUE TO OTHERS? WHAT OTHER BENEFITS CAN BE DERIVED WITHOUT ADDITIONAL INPUT COSTS?

The accreditation package can also support other important functions such as vulnerability reporting, incident reporting analysis, data call responses, the creation of security management products, configuration control and inventory control. As an example, the Air Force implemented the Department of Defense Computer Security Technical Vulnerability Reporting Program (CSTVRP) on 27 Apr 87. The CSTVRP requires the ISSO to identify known vulnerabilities and report them for subsequent action. The ISSO can use the repository of information found in the accreditation data base files to support and minimize efforts in the reporting to the CSTVRP.

Essentially, once a vulnerability has been determined, it could be linked with the appropriate accreditation package describing the network configuration plus all the software and hardware used by the suspect system. Likewise, the data base can be used to determine the location of other possible suspect systems with the same potential vulnerability, thus gaining control of locating the vulnerability throughout a given organization, agency or several agencies. Likewise, at a higher levels of command, the data base can be used as input to a prediction analysis model. Such a model could be designed to locate other possible suspect systems with the same potential vulnerability while actually predicting a possible security incident or break-down (vulnerability wise) in a system before it happens.

ACCREDITATION PROCEDURE

The following is an extract of the actual procedure, published in an ESC regulation, outlining the preparation of an accreditation package.

HOW? All AISs which process, store, transfer, or receive unclassified, sensitive-unclassified, or classified information must be accredited before they may legally be operated in any particular functional area or location. This applies to all systems; government owned, leased, or on loan from other organizations. While accreditation can only be granted by the DAA, interim approval-to-process may be granted by the designees of the DAA.

1. Original Accreditation Process. The accreditation process applies to any AIS processing unclassified, sensitive-unclassified, or classified and requires the submission of an accreditation package for subsequent approval.

2. When to submit an Accreditation Package. The original request should be submitted not later than 60 days prior to desired initial operating capability (IOC) or as soon as the required information is known on specific components, configuration, and interfaces. On large AISs where the purchase contract calls for a critical design review (CDR), submit the package in the development phase immediately after the CDR.

3. Types of Accreditation Requests/Methods of Submission. There are two ways of submitting accreditation packages based upon the requirements.

a. Single Accred ation. The primary method of requesting accreditation is to submit only one AIS per package. The reasons for this type submission vary, but range from the complexity of accrediting a large AIS to the simplicity of being able to manage accountability easier by having only one AIS per package. And there are no restrictions.

b. Type Accreditation. This method permits the submission of one package requesting accreditation of several AISs and all at one time. There are certain restrictions on a "type" submission: All the AISs must be used for the same mission, installed in the same general location, operating in the same security mode, processing the same classification levels, have the same basic hardware configuration, made by the same manufacturer (like all Z-150s), and assigned to the same CCSSO. Do not mix types of AISs within the same package.

4. Types of Approval-to-Process. Once an accreditation package has been submitted, you may receive a "Temporary approval-to-process", "Interim approval-to-process", or "Accreditation".

a. Temporary Approval-to-Process. Temporary approval is a special case usually based upon the requirements to test a research or developmental AIS for a limited time-period such as 30 days, 45 days, etc. The issuance of temporary approval-to-process is based upon the complexity of the AIS and any network connectivity. A package of this nature may be approved by either the DAA or the designees of the DAA after the receipt of the accreditation package and is based upon a complete review of the accreditation package.

b. Interim Approval-to-Process. Interim approval-to-process is the typical first step in the accreditation process. An "Interim" may be granted by the DAA based upon a preliminary review of the accreditation package. Upon review, temporary waivers may be granted, on a case-by-case basis, for the operation of an AIS which has security deficiencies if the waiver supports the time-critical, mission-essential processing requirements of the command. An "Interim" may be issued on any size AIS, networked or standalone, located inside or outside of secure facilities, regardless of the classification level of information (unclassified, collateral or SCI) being processed. It applies to all AISs operating in the

Dedicated, System High, Compartmented, or Multilevel Mode of operation.

c. **Accreditation.** Full accreditation for any AIS can only be granted by the DAA after a site visit and only after a full test of the security controls of the entire system. It applies to any AISs which may have previously been given an interim approval-to-process.

5. **Updating an Accreditation Package.** When certain operational changes are made in an accredited AIS, its accreditation package must be updated or the DAA may cancel the accreditation. Updates are required when:

a. The AIS hardware or software configuration changes at the component level, not board level.

b. The AIS is relocated to another area, building or room.

c. The security mode of operation of the AIS changes.

d. The classification of material processed by the AIS is changed.

e. The AIS is being connected to a network not previously connected.

6. **Rescinding Accreditation.** The DAA may cancel the accreditation of an operational AIS if violations are found in the operational status of the AIS. However, there are acceptable reasons for operational changes that do not normally constitute rescinding accreditation. Accreditation is not rescinded for:

a. Substitution of components while components are in maintenance. However, if the original component is not returned to the AIS when repair is completed, then an update must be accomplished to reflect the current serial number.

b. Relocation of an AIS providing the accreditation package is updated to reflect the relocation and providing the relocation was accomplished IAW established procedures.

c. Addition of new terminals or peripheral devices providing the accreditation package is updated to include the new devices.

7. **Three Year Anniversary Review.** Each accreditation will be reviewed every three years. The ISSO is responsible for ensuring the recertification of each accredited AIS upon its three-year anniversary. If undocumented changes have been made to the AIS an updated package will be sent to the DAA.

8. **The Accreditation Package.** The following accreditation checklist has been automated and is the basis of the dBase III Plus Accreditation Package. It is imperative that the information be accurate and the format strictly followed. It consists of two parts:

a. **Cover Letter.** The first sheet of an accreditation package is a cover letter. It contains a statement by the cognizant certifying authorities that the AIS meets minimum requirements of all security directives, it permits verification by the commander, and reflects the required coordination. It must be signed by the CCSO, the TEMPEST officer, and the Commander. The cover letter is UNCLASSIFIED when removed from the classified accreditation package.

b. **Accreditation Checklist.** The checklist applies to all command AISs whether office information systems, standalone computers, small dedicated AISs, or large mainframes. The checklist is usually classified CONFIDENTIAL when completed, but may range from unclassified to highly classified.

9. **Accreditation Package Accountability.** Two data elements are used accounting and tracking accreditation packages in both manual and automated systems at various command levels. These are covernames and package numbers.

a. **Covername.** The covername is one or more words, no longer than fifteen characters, which is assigned as a local system identification (SYSID). It must NOT relate to the use of or to the name of the AIS. The covername (SYSID) is centered on the top of the coversheet and the accreditation checklist.

b. **Package Number.** The package number is a minimum of four concatenated fields: the MAJCOM (ESC); the unit; a one-up serial number within the year; and, a subordinate unit if applicable. ESC-6914-89001 is an example of the first package submitted by the 6914ESS in 1989. ESC-EST-89001-OLMB is an example of a operating location "OLMB", subordinate unit to a parent unit of ESC-EST. The package number is placed on the top-left corner of the coversheet and the accreditation checklist.

10. **Attachments by Separate Submission:** When the requirement for an attachment to a package exists, it may be sent by a separate transmittal document. Ensure that the document contains the AIS covername and package number. The following are current requirements:

a. **Block Diagrams and Floor Layout Drawings.** The DAA requires block diagrams and or floor layouts on AIS depending upon the complexity of the AIS and its connectivity. The following rules apply:

(1) For personal computers located in an office environment, neither a drawing or block diagram is required.

(2) For Office Information Systems located in an office environment, a block diagram is required.

(3) For large AISs installed within an IPC, a scaled drawing or configuration chart showing the location of the AIS major components is required. Include information on all communication

lines to other computers, networks, and peripherals. Indicate the location of all black phones. Terminals located in the user areas do not have to be shown since their location is already documented in the package.

b. Risk Analysis. AFR 205-16 states that the DAA determines the amount of information needed for issuing approval-to-process on an AIS and outlines the procedures for performing a risk analysis relative to the AIS operating environment in a building. For clarification, "building" includes any building where entry is not controlled by an armed guard on a constant 24-hour-a-day basis and does not have a restricted fence. The following requirements for risk analysis exists:

(1) Any AIS located in a building outside of a secure environment which processes TOP SECRET requires a risk analysis for the AIS itself.

(2) Any AIS located in a building outside of a secure environment which processes SECRET or CONFIDENTIAL requires one risk analysis for all similar AISs within the same office complex within the same building. Any secure facility where entry is controlled by a armed guard and has no restricted area fence are exempt.

(3) An analysis is considered valid until some building configuration change impacts upon its documented security measures.

c. Provide other attachments as necessary. The DAA may require any of the referenced documents in the answers.

SAMPLE ACCREDITATION PACKAGE

The following is an extract of the questions found in an accreditation package. As a sample, it does not contain all the information presented to the DAA. Due to the nature of the operations of ESC, some questions have been eliminated for security reasons. However, the sample is indicative of the detailed documentation presented to the DAA in order to obtain approval-to-process.

ACCREDITATION PACKAGE FOR (Covername)

PACKAGE: (Command)-(Unit)-(Year-Serial)-(Subordinate unit)

1. Unit Identification. Enter the complete address for the organization and location of the activity for which the AIS accreditation is being requested.

2. Information System Security Officer (ISSO). Enter the name (with rank), title, organization/office symbol, phone numbers, and message address of the ISSO who will be responsible for the AIS(s) when operational.

3. **Mission Statement.** Describe the role of the AIS(s) documented in this package in support of the organization's mission. Include a description of any network connectivity requirements.

4. **AIS Identification.** Enter the local system identification, assigned package number, system nomenclature, trusted computing base, number of systems in the package, and whether the system is connected to a local area network and/or external communications. More than one AIS may be included if type accreditation is being requested for standalone AISs or if the package represents a cluster of AISs being networked on its own network device.

5. **Operating System and Commercial Software.** List all software installed and operating on the system(s) in this package. Include all operating system(s), data base system(s), communications software, security software, off-the-shelf software, etc.

6. **Security Mode of Operation.** State the AIS's security mode of operation; Dedicated, System High, Compartmented (Partitioned), or Multilevel, and the kind of processing required; continuous or periodic. If periodic, indicate the time period; hours and number of days per week.

7. **User Clearance Level.** State the formal clearance, formal access, and need-to-know requirements of all users, both direct and indirect, which are anticipated to be authorized access to the AIS(s) contained in this package.

8. **Data Classification Level.** Estimate, in percentages by category, the amount of information which is anticipated to be processed on the AIS(s) in this package. Indicate percentages for: Unclassified, Confidential, Secret, and Top Secret.

9. **Facility Accreditation.** List all buildings and rooms in which all components of all system(s) in this package are located, If located in a secure facility, provide the accreditation authority letter date or message DTG authorizing storage. If the AIS(s) does not process classified and is not located in a secure facility, enter the date, title, and number (if any) of the local-unit physical security policy used to protect the AIS.

10. **Hardware Environment.** List all hardware components installed and operating on the system(s) in this package. For each hardware component enter: the building number, room number, manufacturer, model number, type of component, and serial number.

11. **Audit Trails.** Describe the audit trail procedures used to record data and facilitate review. List the names of the data elements recorded and whether these functions are manual, automated, or a combination of both.

12. **User Identification.** Describe how each user is identified as approved, with an established need-to-know.

13. **User Authentication.** Describe how the AIS authenticates each person attempting access. The mechanism for doing this may include software, hardware, and other measures such as user-identification and passwords, to validate the identity, and file-access authority of the AIS user. Also, describe how user-passwords are generated, disseminated, and controlled.

14. **Product Control.** Describe the labeling procedures for output products, including hardcopy, magnetic and transportable media.

15. **Sanitization.** Describe the sanitization procedures for storage and transportable devices. The statement "IAW (regulation or procedure name, date)" is acceptable.

16. **Risk Assessment.** Has a Risk Assessment been performed on the AIS? If so, enter the date of the assessment, assessment report title or number (if any).

17. **ST&E or Certification.** Has a Systems Test and Evaluation (ST&E) or Certification been performed on the AIS? If so, enter the date of the ST&E/certification, report title or number (if any).

18. **Configuration Management.** Who performs configuration management and under what authority?

19. **Security Education.** What are the security education procedures for the initial training of the AIS users for both using the systems and security procedures? What recurring training is done to continue security awareness?

20. **Emergency Destruction Procedures.** Refer to procedures for emergency destruction in the event of evacuations, natural disasters, hostile actions, etc., to prevent the compromise of sensitive information.

21. **Contingency Plans.** Refer to the contingency plans to be used for file recoveries, systems backups, auxiliary power, off-site storage of critical materials, etc.

22. **Previous Security Incidents.** If a reaccreditation of an AIS, describe any security incidents or spillages experienced in the previous three years.

23. **Maintenance.** Provide a complete summary of the maintenance procedures and state the clearance level of all contract personnel.

24. **Network - Host Connectivity.** List all network connectivity being utilized by the system(s) in this package. For each host connection enter: the host name, the reason for the connectivity (e.g., bulk data transfer, electronic mail, remote query, data base update, etc), and the classification level of the connectivity.

CONCLUSION

The data base structures enable the DAA and other users to produce many reports: management products by organization; by facility; by type of equipment; by types of software; by classification level; by security mode; and, many others.

An automated accrediiation package is a great security tool. As the ESC DAA, I am promoting the use of an automated accreditation package, but I do not believe that the promotion of such a security tool should simply stop within ESC. It is needed throughout the Department of Defense and the government. It tends to solve other problems such as host-tenant agreements which often require different accreditations for the different services, sometimes duplicative. I am convinced that its use is just beginning to unfold and its development could potentially be one of the "most significant" impacts on ISS for many years.

A STRUCTURED APPROACH TO RISK ASSESSMENT:

AN INNOVATIVE CONCEPT

(DESCRIBED BY A CASE STUDY OF THE CONCEPT AS APPLIED TO THE DEPARTMENT OF ENERGY'S RISK ASSESSMENT REQUIREMENTS)

June 30, 1989

Submitted in Response to the 12th NCSC Call for Papers

Topical Category: Management & Administration
(Managing Risk)

Prepared by:

Jennie A. Stevens
&
Richard E. Weiner
Booz, Allen & Hamilton Inc.
4330 East West Highway
Bethesda, MD. 20814
Office Tel. No. (301) 951-2071

A STRUCTURED APPROACH TO RISK ASSESSMENT: AN INNOVATIVE CONCEPT

Prepared by:

Jennie A. Stevens & Richard E. Weiner
Booz, Allen & Hamilton Inc.
Bethesda, Maryland

ABSTRACT: The purpose of this paper is to present a newly developed concept for computer security risk assessment that was developed in 1988 and 1989 by Booz, Allen & Hamilton Inc. The concept, when effectively applied to an organization's risk assessment needs, provides significant cost savings, promotes management involvement, and provides a framework for performing non-labor intensive updates over the life-cycle of a given system. In order to illustrate the application of this new concept to a real world situation, the paper examines a case study of its application to the Department of Energy's risk assessment needs. While it should be noted that the concept must be customized to an organization's "culture," way of doing business, etc., it has broad utility and application to a majority of organizational types: governmental (federal or local), commercial, and international. Using our concept risk assessment guidelines and the actual performance of risk assessments can be readily adapted and provide a cost effective way to assure high levels of computer systems security. Booz, Allen is currently investigating development of structured approaches for use in preparing contingency plans and security plans, as well as for guidance to support the complicated, often poorly-executed processes of certification and accreditation.

1. INTRODUCTION: AN OVERVIEW OF THE BOOZ, ALLEN CONCEPT

Booz, Allen & Hamilton, under contract to the Department of Energy's Office of ADP Management and the Computer and Technical Security Branch, recently developed improved risk assessment guidance for use by DOE and DOE Contractor organizations. The new tool is entitled "The DOE Risk Assessment Guideline -- A Structured Approach, and was developed based upon Booz, Allen & Hamilton' concept.

The concept is, in essence, the framework upon which an individual organization's customized guideline is built. The concept provides a systematic structure and approach to the various evaluations and searches for information. It simplifies the risk assessment process by recognizing existing security initiatives and providing much of the necessary data and decision-making processes that comprise a risk assessment. The intent of the guideline is to provide, in one package, all information necessary to conduct and record the results of a risk assessment. Through application of this concept, in addition to required documentation, a useful end-product -- an Executive Summary -- results. The concept can be applied to develop a comprehensive guideline for most organizations in the government and private sectors.

The approach consists of six specific steps, each of which is guided by specific instructions and special worksheets. The worksheets are supported by informational resource tables. The worksheets solicit specific types of information necessary to support an organization's risk assessment process. The resource tables provide the majority of data and information necessary to complete the worksheets. Data sets provided in the resource tables are customized to suit the needs and "culture" of the organization.

The concept also allows the user to go "off-line," if desired, to use any risk assessment tools that have proven useful in the past, and to enter the results of such off-line analyses in the appropriate section(s) of the Executive Summary. Further, the concept encourages the use of an organization's other, already available computer security documentation as input to the process or as supporting documentation. (Such existing documentation might include inventories, security plans, threat statements, etc.). Finally, the Executive Summary, supported by the completed worksheets produced as an end-product of this approach has multiple utility. This is because it partially or fully addresses many areas that are covered during security inspections, compliance reviews, audits, certifications, accreditations, etc.

Use of this structured approach can greatly expedite and simplify the process of risk assessment. It allows those responsible for computer security to develop a comprehensive, sound assessment without the wheel-spinning and dollar waste of many of the currently used methods. Further, its use by a Government agency can provide the benefits of: significantly lower agency expenditures in terms of manpower; greater accountability by program staff; enhanced computer security awareness and training; and a greater assurance for management that the required process is being completed in a consistently meaningful and effective manner.

2. THE DOE CASE STUDY

There were two fundamental objectives that were established at the start of the DOE project: 1) to determine the "sense" of those individuals familiar with risk assessment requirements as to their likes and dislikes about the process and available methodologies and 2) to help install a greater appreciation for risk management as a way of doing business through use of the products of this effort.

Security professionals in both the Government and commercial sector were interviewed to identify what "worked" and what didn't "work" with respect to risk assessment. Consensus among the interviewees was strong that risk assessment is NOT beneficial if it is:

Excessively detailed and lengthy -- making it a paper exercise rather than a beneficial management and security awareness process

- Overly quantitative in approach, thus resulting in an end-product that is difficult to interpret (if not useless)

- Not oriented towards the true "bottom-line": "What is it going to cost to fix the problems identified?"

- Did not directly support the budget process

- Not geared to providing a management level "buy-in" for any remedies required or acceptance of any identified risks.

A thorough review with the persons interviewed of their organization's computer security culture, environment, and unique ADP applications was also undertaken. Again, views regarding the utility of risk assessment underscored many of the same concerns as were expressed above. Risk assessment had become a paper process divorced from the management decision-making process. It was also felt to be important to recognize that the risk assessment must be integrated with the management process in order to achieve accountability for accepting a system's current risk profile and/or for allocating additional security resources.

A limited survey of DOE computer security professionals also underscored that several other elements of the risk assessment process were problematic. Most of the DOE professionals surveyed indicated that they had difficulty in determining the scope of a risk assessment; many were unclear about the amount of documentation they should develop to support the process; and a majority of them felt that it was difficult to realistically identify risks to and place a value upon intangible, subjective assets.

Upon completion of the community wide interviews and DOE survey and review, our analysis resulted in the development of a set of comprehensive objectives for the DOE Guideline. Based upon our findings we recommended that the Guideline for DOE should be:

- Simple to understand and use

- Generally consistent with and useful for both unclassified and classified environments

- Cost-effective in terms of preparation of the Guideline and the amount of personnel time that would be required to perform the actual risk assessment

- Self-contained for ease and speed of utilization

- Appropriate for use at various facilities or applications

- An information source and training aid

- Non-labor or time intensive for the user

- Capable of providing accountability and reasonable documentation

- Adaptable for use/integration with currently used risk assessment methodologies (software and documentation type)

- Flexibly structured to permit use of existing computer security related documentation as input into the risk assessment framework (i.e., inventory, cost data, threat statements, etc.)

- Useful in providing assessments and recommendations of value to managers responsible for accepting risks or planning and funding computer security improvements

- Supportive of the budget and initiatives justification processes.

The approach to conducting risk assessments as would be suggested by this Guideline was developed with these objectives firmly in mind. The Guideline's structured approach fully meets the risk assessment requirements imposed on Federal Agencies and DOE ADP systems by Federal and Department computer security policy. In fact, the need to develop such an approach is given impetus with the publication of OMB Circular A-130, "Management of Federal Information Resources," which places additional emphasis on conducting risk assessments of all types of Government computer systems. Appendix III of A-130 underscored that such assessments are to provide the basis for making informed management decisions related to accepting identified risks or for implementing appropriate cost-effective countermeasures. This is why it was essential for us to develop management involvement and support as an integral element to our approach. A-130 also allows for varied approaches to fulfill the risk assessment requirement: risk assessments may vary from "an informal review of a microcomputer installation to a formal, fully quantified risk analysis of a large scale computer system." This variation consideration is also appropriately handled.

The Department's 1988 publication of DOE Order 1360.2A, Unclassified Computer Security Program, and DOE Order 5637.1, Classified Computer Security Program, also reflect the need to use the risk assessment process as an effective management tool for properly allocating security resources. In fact, the DOE Unclassified Computer Security Program urges those conducting risk assessments to carefully select the risk assessment approach that is best suited to their particular needs: "When used inappropriately (i.e., selecting an inappropriate methodology just to satisfy a general policy requirement), risk assessments can be costly and ineffective for all involved."

3. ORGANIZATION OF THE GUIDELINE

The Guideline may best be visualized as organized into two major parts which can be divided into two separate volumes: (1) Volume I, Guideline Supporting Documentation, which includes general introductions and reference materials and (2) Volume II, Guideline, which is the main body of the Guideline. Volume I consists of Preamble, Foreword, an Introduction, a completed sample, Bibliography, and a Glossary. Volume II consists of the Executive Summary, the 6 steps of the structured approach, including Worksheets and Resource Tables for each step, and a complete set of the Guideline worksheets to be used for copying. Table of Contents are included in both of Volume I and Volume II. The contents and purpose of each of the Guideline's elements if it is assembled as described are as follows:

(1) Volume I of the Guideline:

- INTRODUCTION: The introduction describes the Guideline's background, its underlying philosophy and objectives, and the mechanics involved in using the Guideline. It also provides general instructions for Guideline use. The introduction is meant to provide the user with a brief understanding of how to approach the assessment using the guideline.

- COMPLETED SAMPLE: A completed sample or samples is very important to include since it illustrates how the worksheets and Executive Summary are to be completed. A description of an ADP system/installation is provided, and then the Guideline's approach is used to conduct a risk assessment of this sample system/installation application that is appropriate to the organization.

- ANNOTATED BIBLIOGRAPHY: The annotated bibliography, is a living document covering the period 1983 - to the present, it consists of ten main topical sections on key areas of concern to those conducting risk assessments, from threat and vulnerability-related articles to literature on specific countermeasures for coping with various types of threats. Special interest sections on viruses and networks are also included. In addition, the bibliography contains references to numerous U.S. Government computer security guidance documents. This is an important element of the guideline's utility as an information source and training aid.

(2) Volume II of the Guideline:

- VOLUME II INSTRUCTIONS: The overview of the risk assessment process summarizes the instructions that will be followed to perform the risk assessment, and includes a "fan-out" chart showing the risk assessment steps, and identifies the elements of the Guideline that supports the step (e.g., worksheets and resource tables). The "fan-out" chart also shows the relationship between the steps.

- EXECUTIVE SUMMARY: The Executive Summary provides a 6 page set of summary sheets for use in recording the results of each step of the risk assessment, and for obtaining management sign-off for the end-results and any resulting recommendations. This summary can be useful at a Headquarters level for keeping track of the risk assessment requirements.

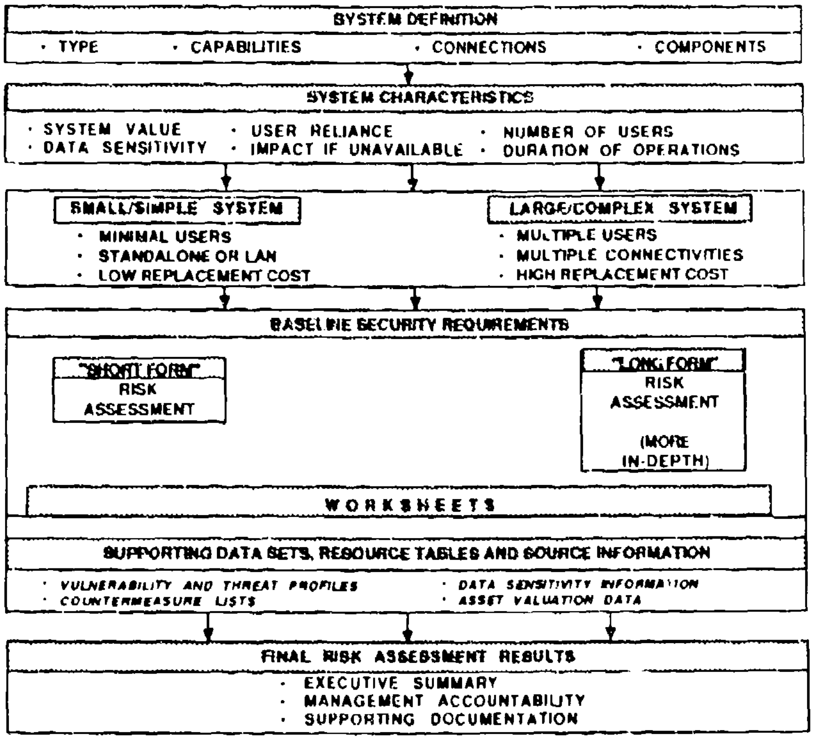
- 6 STEP APPROACH: The 6 steps provide the structured approach for conducting the risk assessment itself. Each step focuses on a particular area of concern; the Resource Tables and Worksheets that accompany each step provide the necessary data sets in an organized format to address each of the areas of concern. Exhibit 1 presents an overview of the 6 steps and their main areas of focus. It also lists the worksheets and resources tables that are used to support each step. A detailed discussion of how the process works -- its mechanics -- is presented in Section 3 below.

- FOR COPYING: The "For Copying" section provides a complete duplicate set of the worksheets used to complete the assessment, and are set aside to promote easy copying and keep the guideline document intact.

EXHIBIT 1

- THE CONCEPT -

APPLYING THE BOOZ, ALLEN CONCEPT: AN OVERVIEW



GLOSSARY: The glossary provides a useful compendium of terms common to the risk assessment process and in use by a particular agency. For example, three DOE sources were used as a starting point for developing the glossary. These were then edited to suit the needs of the Guideline and additional relevant terms were added to ensure coverage of all key terms mentioned herein.

4. DESCRIPTION OF THE GUIDELINE'S MECHANICS

The Guideline provides a systematic, structured approach to the various evaluations and decision making processes that comprise a risk assessment. The intent is to provide an approach that allows you -- whether your system or application is in an unclassified or classified environment or whether it is a PC or large system -- to readily identify and, wherever possible, have available in one package, all information necessary to conduct a risk assessment. The Guideline may be applied to existing or planned systems.

(1) STEP 1: DEFINE YOUR SYSTEM. The purpose of Step 1 is to produce a general definition of your system by looking at several key system features: composition, connections, size, cost(s), and back-ups. There are several uses for this description including the documentation for future analysis and as a source of information to evaluate system importance. First, the current configuration of your system is established to ensure that you have fully identified all major system components and connections. Use of a system configuration diagram provides you a visual opportunity to record and review your system's current configuration. It also allows you to visualize potential vulnerabilities that may exist as a result of your system's connections, data flows, and physical attributes. Second, Step 1 helps you in developing a general cost estimate for your system so that you are able to appreciate how much it would cost to replace valuable components or the entire system. It is also important to have a general appreciation for the cost of your system in order to decide which countermeasures, if any, are justified based on system cost. Step 1 also reviews the type of software and data used by your system, with the objective of understanding approximately how much labor went into the development of each and whether back-ups are available and necessary.

The end products from Step 1 are: (1) A system configuration diagram which depicts your system's major components and connections, (2) a current listing of your system's major components, and (3) rough cost estimates for replacing your system's hardware, software and data.

(2) STEP 2: CHARACTERIZE YOUR SYSTEM, SOFTWARE, AND DATA. The purpose of Step 2 is to characterize your total system in terms of several key characteristics. Questions in two primary areas are answered in this Step: (1) Does your system process any classified information or sensitive unclassified information? If so, what types/levels? Responses to these questions provide the basis for selecting what type(s) of security precautions (countermeasures) are required for your system, software and data; and (2) How

important is your system, its operations, software and data to its users and their organization? Responses to this second question will help you determine the relative importance of the system, software, and data, and provide the basis for determining or validating your contingency planning needs.

The end-products produced in Step 2 are: (1) An assessment of the relative importance of your system, software, and data to their users and organization; and (2) an identification of what types of information you are processing (e.g., unclassified, sensitive unclassified, or classified).

(3) STEP 3: REVIEW BASELINE SECURITY REQUIREMENTS (BLSRs) AND IDENTIFY THOSE NOT MET OR PARTIALLY MET. This is by far the most critical step when applying and adapting our concept to a particular agency or business. The purpose of Step 3 is to determine whether your system's hardware, software, and data -- as they exist today in their current operating environment and utilized by you and your organization -- meet the minimum Baseline Security Requirements (BLSRs) set forth in all applicable orders, guides, or procedures. We perform a careful security requirements analysis so that we may restate Baseline Requirements in readily understandable terms and within known security categories. Once rationalized in this fashion, an analyst performing the risk assessment will be able to quickly discern if the system facility or application under review is protected according to established requirements. It is important to note that this process allows for the recognition of security initiatives that may already be taking place for other reasons that solve computer security related problems. In the previous step you identified whether your system was involved in sensitive unclassified or classified processing. In this step, you are asked to review brief lists of security countermeasures (baseline security requirements) that MUST be in place.

Step 3 will result in an assessment of your current security profile in terms of: (1) whether you currently have met the minimum baseline security requirements that apply to sensitive unclassified and classified ADP processing; (2) a list of any noted deficiencies that must be corrected; and (3) target dates for correcting them. It also allows you to note any areas where you desire to supplement the countermeasures currently in-place if you feel it is justified based on Step 1 and Step 2 results.

Further, for the majority of small/simple systems (as defined in Step 1 of this process), the Step 3 results provide an adequate assessment of the current risks to your system. Therefore, Step 3 also documents the decisions made to accept or upgrade your current risk profile, and provides the basis for obtaining management sign-off for these decisions. For these small/simple systems, the risk assessment process is complete.

(4) STEP 4: REVIEW THREATS AND VULNERABILITIES AND IDENTIFY ANY WHICH AFFECT YOUR SYSTEM. The purpose of Step 4 is to conduct a more extensive review of the threats that might affect your system's hardware, software and data through exploitation of

specific vulnerabilities in your system and its operating environment. In this step, you are asked to record from existing reviews and/or provided worksheets which specific threats could impact your system due to existing deficiencies in your security profile. Further, the Step also addresses the likelihood that a given threat could arise at your site or in your locality. (An uncomplicated probability scheme is provided for your use in order to accomplish this.) Finally, the Step also allows you to specify the priority in which the identified threat(s) should be treated.

The end-products that result from Step 4 are: (1) a threat and vulnerability analysis of your system, facility, and its assets within its operating environment. It will also (2) allow you to identify which of the applicable threats are: very likely to occur, likely to occur, or unlikely to occur. Finally, Step 4 will provide the basis for determining which vulnerabilities should be corrected, and in what order, based on the simple probabilities identified for threat occurrence.

(5) STEP 5: REVIEW AND SELECT COUNTERMEASURES OR ACCEPT CURRENT RISK PROFILE. The purpose of Step 5 is two-fold. It provides an opportunity to review appropriate countermeasures in each of the security discipline areas (the same areas as the BLSRs were sorted) and decide which ones are appropriate for implementation to counter the threat impacts identified in Step 4. However, if your review of the threat impacts does not result in the identification of any new concerns, and confirms that your security program fully treats all possible threat scenarios for your system and site, then Step 5 also allows you to acknowledge this by accepting your current risk profile.

Step 5 results in (1) a prioritized list of countermeasures for implementation in each of the security discipline areas; or (2) a formal acceptance of your current risk profile based on a documented review and analysis of possible threat impacts to your system.

(6) STEP 6: PROVIDE FOR ACCOUNTABILITY AND OBTAIN REVIEW: MANAGEMENT UNDERSTANDING OF YOUR RISK PROFILE AND COUNTERMEASURES REQUIRED. Step 6 is the last and final step in the risk assessment process. It is a highly critical step, one that is often overlooked or neglected. The purpose of Step 6 is to obtain management review and provide accountability for the decisions and choices made throughout the risk assessment process. It provides a mechanism for briefing, reviewing, and discussing the risk assessment results with management and planning for resources required for implementing the countermeasures identified. This provides a mechanism to help management in the budget/justification process by providing a readily understandable and defensible approach to choosing countermeasure initiatives.

The Executive Summary Block for Step 6, Obtain Accountability: Management Understanding of Your Risk Profile and Countermeasures Required, provides a sign-off area for management to review the results of the risk assessment, and accept the current risk profile. There is an area on this form that provides for comments to elaborate on any special reasons for particular choices. This sign-off is the final end-product.

5. SUMMARY REMARKS

We believe that our development of the concept described above and its ongoing refinement is making an important contribution to the computer security community for several significant reasons. Foremost, it simplifies and makes more logical a heretofore very frustrating and time-consuming process. No less important, however, is its place in reducing the inordinate waste of federal (tax-payers) or corporate dollars on repetitious, inconclusive assessments. Hopefully those organizations who will apply the concept to their system risk assessment needs will share this belief. The "Bottom Line" of this process is that you are provided with carefully documented recommendations for countermeasures based upon identified requirements that are not being met. It is for only these requirements that are not met that it is necessary to search for related threats or vulnerabilities. Key to the process is that you only need to analyze what is necessary and do not have to perform complex assessments to prove you are already satisfying many security requirements.

LAVA'S DYNAMIC THREAT ANALYSIS

Suzanne T. Smith
Los Alamos National Laboratory
Safeguards Systems Group, MS-E551
P. O. Box 1663
Los Alamos, New Mexico 87545

Introduction

LAVA (the Los Alamos Vulnerability/Risk Assessment system) is an original systematic approach to risk assessment developed at the Los Alamos National Laboratory to deal with risks inherent in massive, complicated systems. Characteristics of such systems are huge bodies of imprecise data, indeterminate (and possibly undetected) events, large quantities of subjective information, and a dearth of objective information. The impetus for developing LAVA was the existence of Federal requirements for periodic risk assessments of a variety of systems, coupled with the need for an inexpensive, reusable, automated risk assessment tool firmly rooted in science [1]. When the LAVA project began in 1983, there was no such tool [2]; LAVA was designed to fill that gap [3].

LAVA is an alternative to existing quantitative methods, providing an approach that is both objective and subjective, and producing results that are both quantitative and qualitative. In addition, LAVA could be used as a self-testing aid in preparing for inspections, as a self-evaluating device in testing compliance with the various orders and criteria that exist, and as a certification device by an inspection team.

LAVA is a three-part systematic approach to risk assessment that can be used to model a variety of application systems such as computer security systems, communications security systems, information security systems, and others. The first part of LAVA is the mathematical model based classical risk assessment [4,5], hierarchical multilevel system theory [6,7], decision theory [8-11]. fuzzy possibility theory [11-14], expert system theory [15,16], utility theory [17,18], and cognitive science [19,20]. (The mathematical model has been presented at other technical meetings [21-23], and generally will not be addressed in depth in this paper.) The second part is the implementation of the mathematical risk model as a general software engine, written in a commercially available programming language for a large class of personal computers. The third part is the application data sets written for a specific application system. LAVA provides a framework [24] for creating applications upon which the software engine operates; all application-specific information appears as data.

We use the LAVA system to develop a hierarchical structure and sets of fuzzy analysis trees for modeling risk assessment for a variety of systems associated with computer and information security. With LAVA, we build knowledge-based expert systems to assess risks in application systems comprising a subject system and a safeguards system. The subject system model is sets of threats, assets, and undesirable outcomes; because the threat to security systems is ever-changing, LAVA provides for an analysis of the dynamic aspects of the threat spectrum--the dynamic threat analysis [25] is the subject of this paper. The safeguards system model has three parts: sets of safeguards functions for protecting the assets from the threats by preventing or ameliorating the undesirable outcomes; sets of safeguards subfunctions whose performance determines whether the function is adequate and complete; and sets of issues, appearing as interactive questionnaires, whose measures (in both monetary and linguistic terms) define both the weaknesses in the safeguards system and the potential costs of an undesirable outcome occurring.

The user need have no knowledge of formal risk assessment techniques. All the technical expertise and specialized knowledge are built into the software engine and the application system. LAVA applications include the popular computer security application [26-29] and applications for nuclear power plant control rooms [30], embedded systems, survivability systems, transborder data flow systems [31], property control systems, nuclear processing plant safeguards systems [32], and others. LAVA application systems have been in use by Federal government agencies since 1984.

LAVA Application Models

The General LAVA Application Model

Using LAVA, we build knowledge-based expert systems for assessing risks in applications systems. There are two parts that define an application model. The first part is composed of the following elements: the hierarchical structure and trees that define the framework of the model--the threat, asset, and outcome sets; the fuzzy outcome possibility matrix; the safeguards functions for each threat-asset pair, based upon the kinds of interactions that might result in one or more of the outcomes; the safeguards subfunctions for each function; mitigating factors for outcome severity; and the contributing factors, both linguistic and monetary, to the potential cost of a successful attack. The second part is the set of questionnaires, implemented as data sets on which the general software engine operates: the vulnerability assessment questionnaire, the outcome severity mitigation questionnaire, the dynamic threat questionnaire (if applicable), and the monetary and linguistic impact (or cost) questionnaires.

The vulnerability assessment questionnaire for a given application is concatenated from a library of category questionnaires that come from specific security orders, inspection criteria, interviews with various experts in the field, and general good security practice. The questions themselves represent individual safeguards (called "safeguards elements") or portions of safeguards (called "safeguards attributes") that are related through a database structure to one or several of the safeguards subfunctions. The vulnerability questionnaire can comprise from a few hundred to several thousand questions, depending on the required analytical depth.

The other questionnaires are all considerably smaller than the vulnerability questionnaire. The outcome severity mitigation questionnaire inquires about the presence and estimated effectiveness of any mitigating situations that might be pertinent. If intelligence information is available and analytical detail about the dynamic threat is required, the dynamic threat questionnaire seeks information about the motivation, capability, and opportunity of the current known threat and about the attractiveness of each asset set to the threat; if such information is not available, the user estimates a relative attractiveness factor for the asset sets and whether the dynamic threat is the same as or, in varying degrees, larger or smaller than the background (static) threat. The impact questionnaires ask cost-related questions in either linguistic or monetary terms. With the exception of the intelligence-based dynamic threat questionnaire, all of the questions in these questionnaires number in the single or double digits (usually not more than a dozen or so questions).

Users are not required to be expert risk analysts to use a LAVA application--that mathematical and analytical expertise already exists as a part of the mathematical model and its general software engine. Expert knowledge about the structure and characteristics of safeguards and security systems is a part of the specific application model. The only knowledge required of users is information about that which they know best: their own facility, organization, assets, equipment, policies, procedures, and security practices. The LAVA software system elicits this information by means of the automated questionnaires administered to evaluation teams whose members have diverse backgrounds and responsibilities. LAVA generates both general reports for management and detailed reports for operations staff from information obtained from the questionnaires.

LAVA/CIS: The Computer/Information Security Model

For our computer/information security application model, LAVA/CIS, we postulate four assets: 1) the facility, including physical plant and personnel; 2) hardware, including all computing and ancillary pre- and post-processing hardware; 3) machine-interpretable information, including software, input and output files, and databases; and 4) human-interpretable information, including documents, screen displays, graphs, charts, film output, and so forth. The model's threat set consists of three threats: 1) natural, random, and environmental hazards; 2) direct or onsite humans, including the authorized insider; and 3) indirect or offsite humans. Figures 1-2 show the hierarchical structures for two of the threat categories with respect to the four asset categories; included in these hierarchies, and discussed later in this paper, are representative safeguards functions and subfunctions associated with each threat-asset pair. Figure 3 shows how this relates to the entire model.

There are six undesirable outcomes considered in the computer/information security model: 1) unauthorized access or use; 2) modification or tampering; 3) damage or destruction; 4) theft; 5) unauthorized disclosure; and 6) denial of use. It is important to note that a single event can result in the simultaneous occurrence of more than one of the outcomes. Figure 4 shows the outcome possibility matrix for the threat-asset combinations; a value of zero indicates that the outcome is impossible for that threat-asset combination, and a value of unity means the outcome is possible for that threat-asset pair; greater granularity can be achieved by assigning values lying between zero and unity.

Once we have established the threat, asset, and outcome sets and the outcome possibility matrix, we then address what constitutes the ideal safeguards system for preventing the threats from attacking the assets and achieving the postulated outcomes. For this we define a set of safeguards functions for each of the distinguishable threat-asset pairs (nine T-A pairs, in this application) in such a way that the relative importance of each function within the set of functions for each T-A pair is about the same. Then, for each of the individual safeguards functions, we define a set of subfunctions that provide performance criteria for the adequacy and completeness of that safeguards function; each of the subfunctions is devised so that the relative importance of each subfunction within a specific function is about the same. Again, Figs. 1-3 show the safeguards functions and subfunctions for each distinguishable threat-asset pair.

The Dynamic Threat Analysis

Both government and corporate organizations may be the targets of a variety of hostile agents [33,34], and the intensity of the threat may change with time and circumstances. The dynamic threat strength can be analyzed if the subject system is extremely sensitive to a changing threat and if the subject organization has access to the kinds of information the analysis requires. The dynamic threat analysis takes into account possible threat agents and their potential attack goals with respect to the target(s) of the attack.

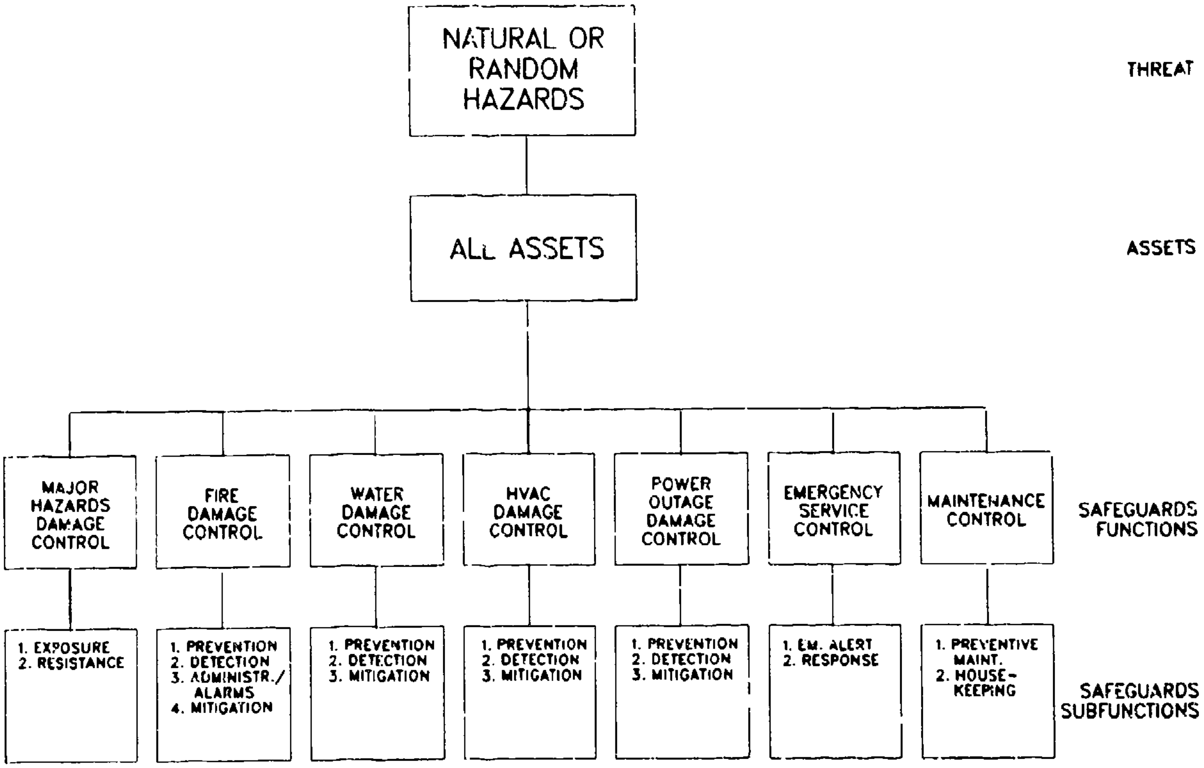
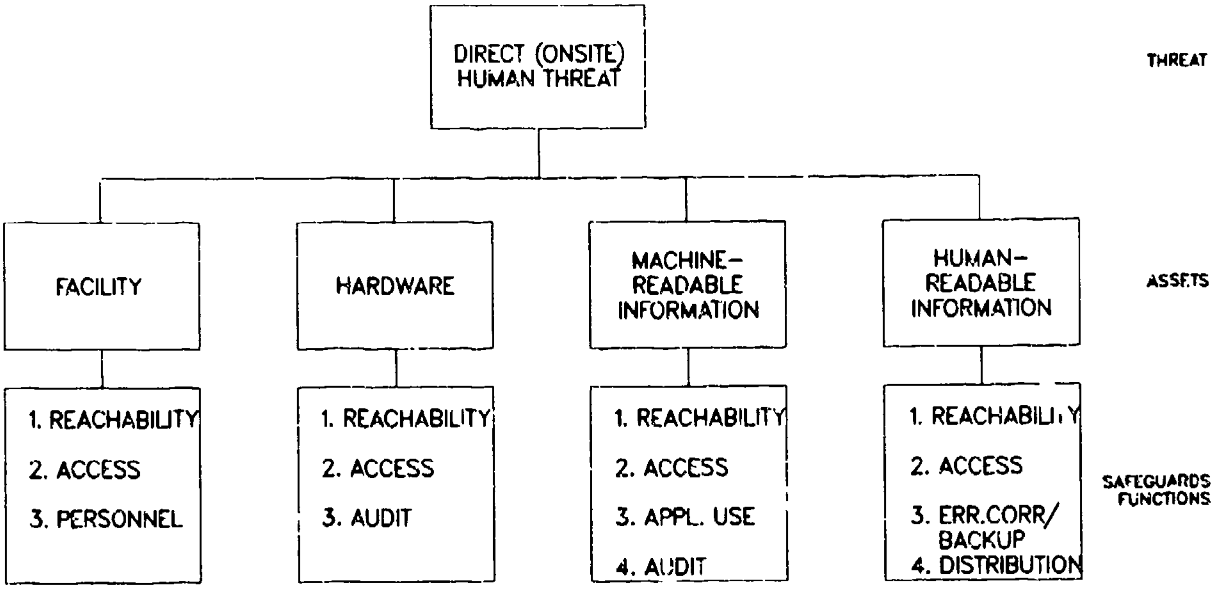

NATURAL OR RANDOM HAZARDS
THREAT
ALL ASSETS
ASSETS
MAJOR HAZARDS DAMAGE CONTROL
FIRE DAMAGE CONTROL
WATER DAMAGE CONTROL
HVAC DAMAGE CONTROL
POWER OUTAGE DAMAGE CONTROL
EMERGENCY SERVICE CONTROL
MAINTENANCE CONTROL
SAFEGUARDS FUNCTIONS
1. EXPOSURE 2. RESISTANCE
1. PREVENTION 2. DETECTION 3. ADMINISTR./ ALARMS 4. MITIGATION
1. PREVENTION 2. DETECTION 3. MITIGATION
1. PREVENTION 2. DETECTION 3. MITIGATION
1. PREVENTION 2. DETECTION 3. MITIGATION
1. EM. ALERT 2. RESPONSE
1. PREVENTIVE MAINT. 2. HOUSE-KEEPING
SAFEGUARDS SUBFUNCTIONS

Fig. 1. Natural hazards hierarchy for computer/information security application.


DIRECT (ONSITE) HUMAN THREAT
THREAT
FACILITY
HARDWARE
MACHINE-READABLE INFORMATION
HUMAN-READABLE INFORMATION
ASSETS
1. REACHABILITY 2. ACCESS 3. PERSONNEL
1. REACHABILITY 2. ACCESS 3. AUDIT
1. REACHABILITY 2. ACCESS 3. APPL. USE 4. AUDIT
1. REACHABILITY 2. ACCESS 3. ERR.CORR/ BACKUP 4. DISTRIBUTION
SAFEGUARDS FUNCTIONS

SAFEGUARDS SUBFUNCTIONS BRANCH FROM EACH SAFEGUARDS FUNCTION.

Fig. 2. Direct (onsite) human threat hierarchy for computer/information security application.

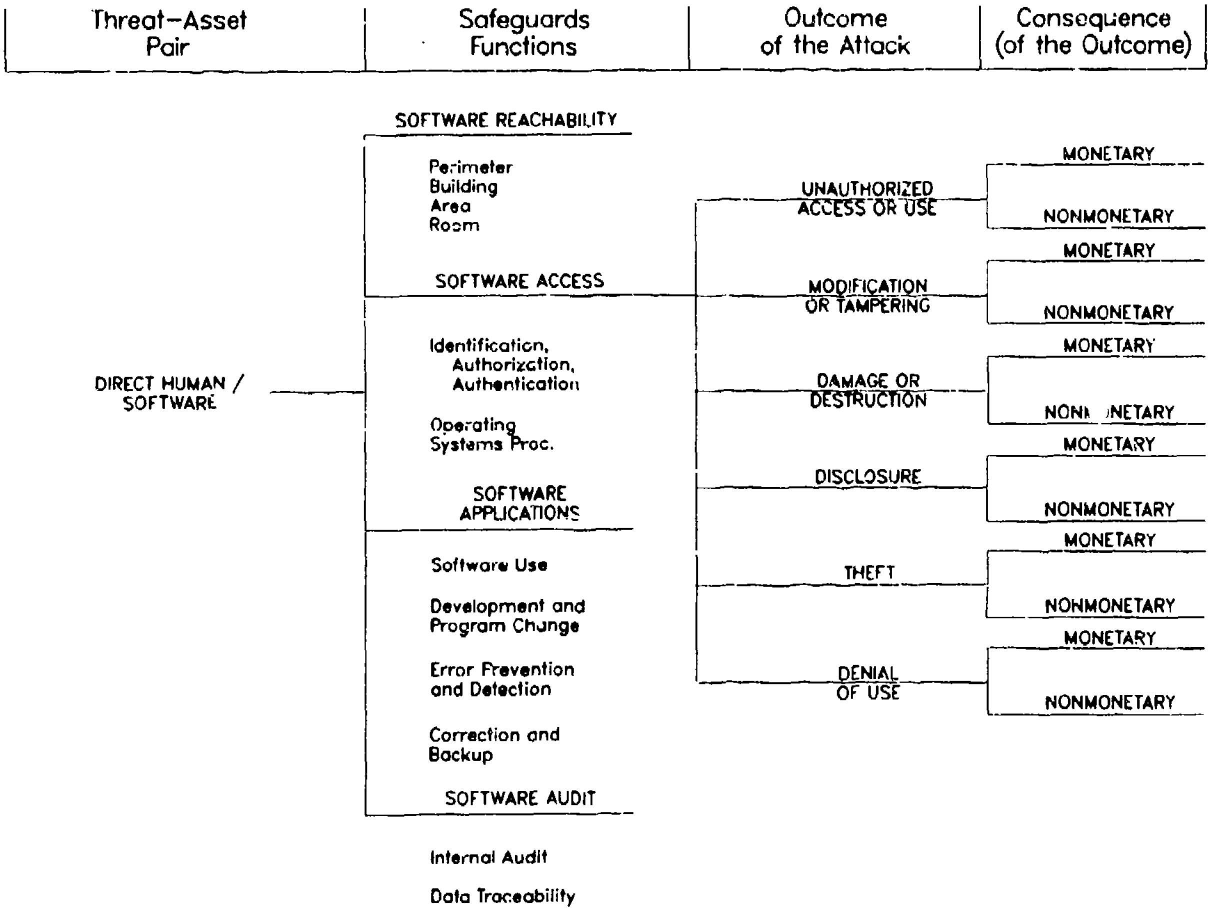


Fig. 3. Direct human/software scenario analysis tree.

	Unauthorized Access or Use	Modification or Tampering	Damage or Destruction	Disclosure	Theft	Denial of use
Natural Hazards – Facility	0	1	1	0	0	1
Natural Hazards – Hardware	0	1	1	0	0	1
Natural Hazards – Software	0	1	1	0	0	1
Natural Hazards – Documents/ Displays	0	1	1	0	0	1
Direct Human – Facility	1	1	1	1	1	1
Direct Human – Hardware	1	1	1	1	1	1
Direct Human – Software	1	1	1	1	1	1
Direct Human – Documents/ Displays	1	1	1	1	1	1

Fig. 4. Outcome possibility matrix for computer/information security application.

The threat component measures the relative strength of identifiable threat agents in terms of asset attractiveness, motivation, opportunity, and capability with respect to the spectrum of assets, the corresponding safeguards functions, and the set of possible outcomes. Asset attractiveness to the threat agent is different from asset value to the organization, reflecting the different value structure of the threat agent; it is a rough indicator of attack likelihood in that a threat agent is unlikely to mount an attack on an unattractive asset. Motivation is a measure of how much effort or what part of his resources a threat agent is willing to expend on an attack and how dedicated he is to carrying out the attack. Capability is a measure of the resources--knowledge (training), information (intelligence), funds, skills, equipment, armament, personnel--the threat agent has at his disposal. Opportunity is a measure of how easy it is for the threat agent to achieve an enabling proximity for an attack: how easy it is for him physically to reach the object of attack, how easy it is for him to attack or to access the object, how easy it is for him to travel undetected (both in the neighborhood of the object of attack and from afar to get near the object), and so forth. Opportunity is separate and different from potential site vulnerabilities. Figure 5 illustrates the analysis structure for the dynamic threat analysis.

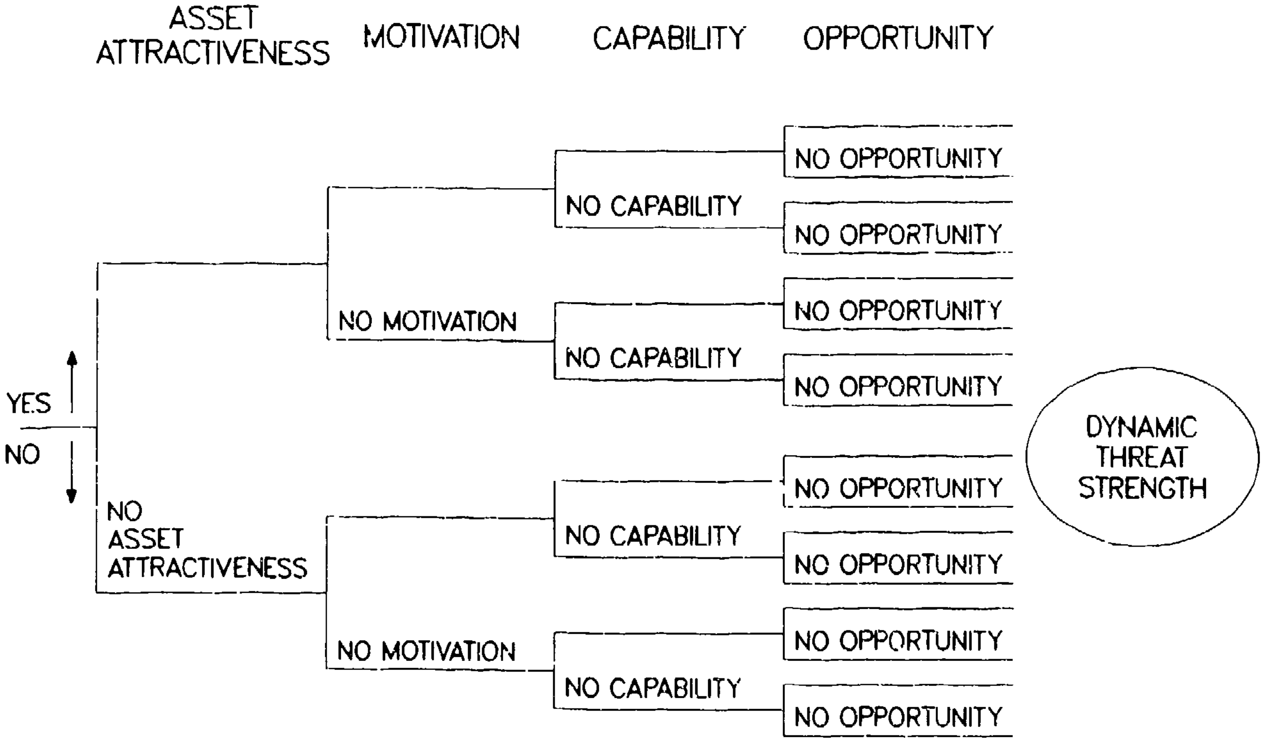


Fig. 5. Analysis structure for dynamic threat.

There are several broad categories of threat agents having a variety of goals. Possible categories of threat agents might be, for example:

a) information gatherers (e.g., spies or hostile intelligence services),
b) terrorists,
c) pro- or anti-"X" radicals or extremists (where "X" could be almost anything!),
d) representatives of organized crime,
e) other criminals (non-malicious criminals and pranksters),
f) insiders (employees, contractors, etc.),
g) outsiders with access, and
h) Mother Nature.

The dynamic aspects of the natural hazards may or may not be of interest; these include both random natural hazards, such as volcanic eruptions or earthquakes, as well as the natural hazards more cyclic in nature, such as hurricanes, tornadoes, torrential rains, and the like. The human threat agents in each of these categories all act for different reasons, so they may differ widely in motivation, capability, and opportunity. Similarly, the goals of the attacks may vary, but all categories of goals may be used by all categories of threat agents. Some possible goal categories are

1) information and/or material collection (e.g., espionage or theft of nuclear materials),
2) sabotage,
3) theft, embezzlement, fraud--generally for monetary gain,
4) damage or destruction,
5) extortion,
6) disrupting business or mission, and
7) surmounting an intellectual challenge.

Clearly, more than one of the categories may be the goal of a single attack, and a single attack may be perpetrated by more than one category of threat agent.

The approach to assessing the dynamic part of the threat component by considering categories of threat agents and possible categories of attack goals is parallel to the approaches used for both the vulnerability analysis and the general consequence analysis. Potential scenarios are modeled implicitly as the relationship between the threat-asset pairs and the safeguards functions in the vulnerability analysis, and as the relationship between the assets and the threat elements (asset attractiveness, motivation, capability, and opportunity) in the threat assessment. Similarly, the attack goals are modeled implicitly in the capability component of the dynamic threat measure and are approximately equivalent to the outcomes used in the consequence analysis.

An interactive questionnaire models the contributors to the dynamic threat in terms of specific threat groups. A fuzzy degree of strength is calculated for each group based on asset attractiveness, motivation, capability, and opportunity relative to a specific [threat, asset, safeguards function, outcome] quadruplet. A relational database keeps track of which threat groups can affect each quadruplet so that an overall or total value for the dynamic threat strength can be calculated for each quadruplet, which is used subsequently in the loss exposure calculations.

Conclusions

LAVA's capability to assess the dynamic aspects of the threat spectrum makes it an ideal tool for modeling applications of interest to the intelligence and military communities. It would also be highly applicable in the business community in situations ripe for industrial espionage.

Using the LAVA approach for risk assessment has benefits that do not accrue from the use of other methods. First, the automated report generators produce results that are immediately usable, both to managers who must make major, far-reaching decisions and to the security personnel in the field whose job it is to maintain an acceptable level of safeguards. Second, because LAVA produces both qualitative and quantitative results, users feel more comfortable with the results because they understand both the results and the information that produced those results. Third, because LAVA does not require the user to generate probabilities (often unfounded) for its operation but instead relies on a natural-language user-friendly interface to acquire its data, users are more willing to act upon its results. Fourth, LAVA includes a way to assess the changing, or dynamic, aspects of the threat spectrum. And finally, because of the team environment in which an assessment is performed and discussions that arise among team members, using a LAVA application has proved to be an experience that both raises the security consciousness of the users and enhances the overall working environment at the facility.

References

[1] S. Katzke, "National Bureau of Standards Perspective on Risk Analysis: Past, Present, and Future," presented at the 1st Federal Risk Analysis Workshop, Montgomery, Alabama, January 1985.

[2] S. T. Smith, "A Government-Wide Overview of Risk Analysis Methodologies," presented at the 8th DOE Computer Security Group Conference, Richland, Washington, April 16-18, 1985.

[3] S. T. Smith and J. J. Lim, "An Automated Procedure for Performing Computer Security Risk Analysis." in Proceedings 6th Annual ESARDA Symposium on Safeguards and Nuclear Material Management, 1984, ESARDA 17, pp. 527-530.

[4] N. J. McCormick, Reliability and Risk Analysis: Methods and Nuclear Power Applications. New York: Academic Press, 1981.

[5] W. D. Rowe, An Anatomy of Risk. New York: John Wiley & Sons, 1977.

[6] M. D. Mesarovic, D. Macks, and Y. Takahara, Theory of Hierarchical Multilevel Systems. New York and London: Academic Press, 1970.

[7] Y. M. I. Dirickx and L. P. Jennergren, Systems Analysis by Multilevel Methods. New York: John Wiley & Sons, 1979, pp. 10-82.

[8] P. C. Fishburn, Decision and Value Theory. New York: John Wiley & Sons, 1964.

[9] R. L. Keeney and H. Raiffa, Decisions with Multiple Objectives: Preferences and Value Tradeoffs. New York: John Wiley & Sons, 1976.

[10] R. Schlaifer, Analysis of Decisions Under Uncertainty. Huntington, New York: Robert E. Krieger Publishing Company, 1978.

[11] R. E. Bellman and L. A. Zadeh, "Decision-making in a Fuzzy Environment," Management Science, Vol. 17, No. 4, pp. B141-B164, December 1970.

[12] A. Kaufmann and M. M. Gupta, Introduction to Fuzzy Arithmetic: Theory and Applications. New York: Van Nostrand Reinhold Company, 1985.

[13] L. A. Zadeh, "Fuzzy Sets as a Basis for a Theory of Possibility," Fuzzy Sets and Systems, Vol. 1, pp. 3-28, 1978.

[14] C. V. Negoita, Expert Systems and Fuzzy Systems. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1985, pp. 52-58, 74-88, 95-112.

[15] P. H. Winston, Artificial Intelligence. Reading, MA: Addison-Wesley, 1984, pp. 251-288.

[16] R. Jain, "A Procedure for Multiple-Aspect Decision-Making Using Fuzzy Sets," Int. J. Systems Sci., Vol. 8, No. 1, pp. 1-7, January 1977.

[17] P. J. H. Schoemaker and C. C. Waid, "An Experimental Comparison of Different Approaches to Determining Weights in Additive Utility Models," Management Science, Vol. 28, No. 2, February 1982.

[18] E. M. Johnson and G. P. Huber, "The Technology of Utility Assessment," IEEE Trans. Sys., Man, Cyber., Vol. SMC-7, No. 5, pp. 311-325, May 1977.

[19] L. A. Zadeh, K.-S. Fu, K. Tanaka, and M. Shimura (Eds.), Fuzzy Sets and Their Applications to Cognitive and Decision Processes. New York: Academic Press, 1975.

[20] S. Sudman and N. M. Bradburn, Asking Questions: A Practical Guide to Questionnaire Design. San Francisco: Jossey-Bass, Inc., 1982.

[21] S. T. Smith and J. J. Lim, "An Automated Interactive Expert System for Evaluating the Effectiveness of Computer Security Measures," presented at the 7th Department of Defense/ National Bureau of Standards Computer Security Conference, Gaithersburg, Maryland, September 24-26, 1984.

[22] S. T. Smith, J. R. Phillips, R. M. Tisinger, J. J. Lim, D. C. Brown, and P. D. FitzGerald, "LAVA: A Conceptual Framework for Automated Risk Analysis," presented at the 1986 Annual Meeting of the Society for Risk Analysis, Boston, November 9-12, 1986.

[23] S. T. Smith, "LAVA: An Expert System Framework for Risk Analysis", presented at the 1st International Computer Security Risk Management Model Builders Workshop, Denver, Colorado, May 24-26, 1988.

[24] S. T. Smith and J. J. Lim, "Framework for Generating Expert Systems to Perform Computer Security Risk Analysis," Proceedings First Annual Armed Forces Communications and Electronics Association Symposium and Exposition on Physical and Electronics Security, 1985, pp. 24-1 - 24-7.

[25] S. T. Smith, J. R. Phillips, D. C. Brown, and P. D. FitzGerald, "Assessing the Threat Component for the LAVA Risk Management Methodology," presented at the Ninth DOE Computer Security Group Conference, Las Vegas, Nevada, May 6-8, 1986.

[26] S. T. Smith and J. J. Lim, "An Automated Method for Analyzing Computer Security Risk," presented at the Seventh DOE Computer Security Group Conference, New Orleans, April 17-19, 1984.

[27] S. T. Smith and J. J. Lim, "An Automated Method for Assessing the Effectiveness of Computer Security Safeguards," presented at the IFIPS Second International Congress on Computer Security, Toronto, Canada, September 10-12, 1984.

[28] S. T. Smith and J. J. Lim, "LAVA: An Automated Computer Security Vulnerability Assessment Software System (Version 0.9)," Los Alamos National Laboratory document LA-UR-85-4014, December 1985.

[29] S. T. Smith et al., "LAVA for Computer Security: An Application of the Los Alamos Vulnerability Assessment Methodology," Los Alamos National Laboratory document LA-UR-86-2942, 1986.

[30] S. T. Smith and J. J. Lim, "Assessment of Computer Security Effectiveness for Safe Plant Operation," Trans. Am. Nucl. Soc., Vol. 46, pp. 525-526, June 1984.

[31] S. T. Smith, J. J. Lim, and J. Lobel, "Application of Risk Assessment Methodology to Transborder Data Flow," in Handbook on the International Information Economy. Springfield, Virginia: Transnational Data Report, November 1985.

[32] S. T. Smith and R. M. Tisinger, "Modeling Risk Assessment for Nuclear Processing Plants with LAVA," Nucl. Mater. Manage., Vol. XVII (Proceedings Issue), pp. 315-318, June 1988.

[33] N. R. Bottom, Jr., and R. R. J. Gallati, Industrial Espionage: Intelligence Techniques and Countermeasures. Boston: Butterworth Publishers, 1984.

[34] R. Eells and P. Nehemkis, Corporate Intelligence and Espionage: A Blueprint for Executive Decision Making. New York: Macmillan, 1984.

Anomaly Detection: Purpose and Framework

by

G.E. Liepins
MS 6207 Bldg 4500N
Oak Ridge National Laboratory
(615) 576-5238

H. S. Vaccaro
MS E541
Los Alamos National Laboratory

This work has been supported in part by US DOE Office of Safeguards and Security.

ABSTRACT

This paper places anomaly detection of computer use in the framework of overall computer security. A balance of physical security, access security, anomaly detection, misuse detection, and database management is proposed to provide the maximum practical security for computer systems. The fundamental concepts of the anomaly detection module Wisdom and Sense (W&S), including rule representation, rule generation, rule pruning, and evidence combining are presented.

INTRODUCTION

Computer security has become a volatile issue. Misuse needs to be prevented without compromising system performance or user productivity. Traditionally, security has been addressed through physical and access security, with additional sporadic review of audit logs by security officers. However, with increasing hacker sophistication and insider misuse, these measures are no longer sufficient. Physical and access security by themselves cannot fully protect a system from misuse. As a result, an ever increasing responsibility falls on the shoulders of security officers. Unfortunately, security officers are sorely overburdened. A typical audit log of VMS image termination data for 100 users can generate upwards of 20 megabytes of data per week. More detailed data collection could result in as much as a thousand megabytes per week for the same 100 users.

Typically, nearly all system use is appropriate. Any evidence of misuse is generally hidden by large quantities of routine usage patterns. In principle, the difficulty of detecting misuse is eased somewhat by the recent proposed development of misuse detection expert systems, systems that incorporate security officers' knowledge (Denning, 1987; Sebring et. al., 1988). To the degree that computer security experts are able to articulate what constitutes misuse and specify how such misuse might be detected, misuse detection modules could relieve much of the drudgery of audit log review.

Unfortunately, experience in misuse detection is limited; currently known rules are thought to be able to identify only a small fraction of potential misuse. This lack of expertise and experience points to the need for anomaly detection, the detection of usage that is at variance with historically established, appropriate patterns (Clyde, 1987; Denning et. al., 1987; Hansen and Messier, 1986; Lunt et. al., 1988; Lunt, 1988; Smaha, 1988; Tener, 1988; Vaccaro and Liepins, 1989).

This paper introduces the formal framework of the anomaly detection module Wisdom and Sense (W&S) developed at Los Alamos National Laboratory (LANL) (Vaccaro and Liepins, 1989) and discusses it from the perspective of overall computer security. The interrelated roles of physical security, access security, anomaly detection, misuse detection, and database management are briefly reviewed. The three major challenges facing any anomaly detection module are introduced: the need to summarize the vast quantity of historical data, the need to extrapolate from a small sample of the nearly limitless variety of possible computer transactions, and the need to deal with mixed, yet predominantly categorical data.

W&S addresses these issues by generating a "forest" of decision rules together with clustering of continuous data. The generation of the forest of rules is described. Rule representation, generation, and pruning are described. Rule strength, combining evidence, and post processing are further detailed.

BACKGROUND

Computer security anomaly detection identifies unusual transactions. Generally, unusual transactions arise both from appropriate use as well misuse. Indeed, some misuse may actually be common. Presumably, common misuse will already be known to the security officer and could be culled out by a misuse detection module. Much of the remaining misuse will manifest itself as unusual activity. Thus, the legitimate goal of anomaly detection is to filter raw audit data by two to three orders of magnitude without overlooking that misuse which is unusual. A good anomaly detection system balances these two opposing objectives. It lowers the probability that a legitimate transaction needs to be reviewed, and simultaneously provides high assurance that unusual misuse will not pass undetected.

All transactions flagged by an anomaly detection system should be reviewed by a security officer, and those determined to be indicative of misuse entered into a data base. The data base should also summarize how the security officer determined that a flagged transaction represented misuse and what additional information he needed to make this determination. Such a data base would serve to periodically upgrade the misuse detection module of an overall security system.

An overall security system would incorporate physical and access security, an audit data collection capability, an anomaly detection module, a misuse detection module, the security officer, and a database of findings and rules. All transactions would be screened by both anomaly detection and misuse detection modules. Of note should be the separation of the two detection modules; they serve different, yet related, purposes. Misuse detection identifies usage patterns already known to have a high probability of being inappropriate. Anomaly detection applies statistical methods to identify "exceptional" transactions and system's conditions. Subsequent review of the anomalous transactions and conditions could well result in the addition of rules to the misuse detection module.

The components of an overall computer security system are illustrated in Figure 1, below, and are discussed in greater detail in the following paragraphs.

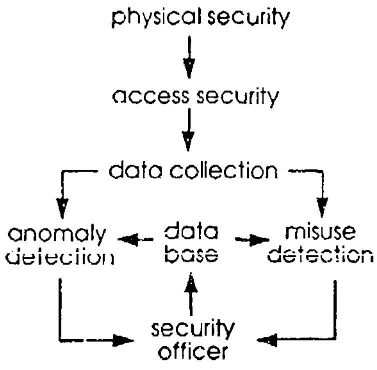


Figure 1. High Level Architecture of an Overall Computer Security System

At the level of detail concomitant with Figure 1., neural networks and statistically based techniques are both advocated for anomaly detection. W&S is one possible implementation of a statistically based anomaly detection system, and operates at the level of individual transactions. On the other hand, a neural network approach is considered to be best suited for anomaly detection in terms of recognition and comparison of keystroke patterns.

Administrative rules, expertly determined rules, and automatically generated rules are suggested to be the constituent components of the misuse detection module, a module whose purpose is to identify those transactions (frequent or infrequent) that represent inappropriate use. Administrative rules are of the form, "No-one but the system's manager should access file X". Expertly determined rules are rules that encode the security officer's knowledge about types of usage patterns that are likely to be indicative of misuse. Automatically generated rules would be generated by machine learning methods (Michalski, Carbonell and Mitchell, 1983 and 1986) from the database of confirmed misuse. This database would be updated as the security officer either independently uncovered misuse or determined that an anomalous transaction was inappropriate.

The separation of the anomaly and misuse modules is somewhat artificial. Analytic tractability, and ease of development and testing of the two modules are two reasons for the separation. Yet at the same time, the separation may help provide an indication of the "holes" in the anomaly detection system, and could bring to light commonly occurring misuse patterns insofar as these would be flagged by the misuse detection module but not the anomaly detection module. Common misuse would signal the need for either review of the counterpart rules in the misuse detection module, or else the need for measures to prevent such misuse. Comparative review between the two modules is desirable, and such a consistency checking capability between modules is being developed in conjunction with W&S.

The data base would include confirmed records of misuse, what auxiliary data has been required by the security officer to resolve anomalies, and several (or more) generations of the anomaly detection rule bases (and possibly misuse detection rule bases). Comparison of rule bases over time would allow the detection of gradual encroachment on the system over time, no one step of which would be individually noted as anomalous. (For example, a user might slowly increase his unauthorized privileges.) As previously stated, this data base would be used to update the intrusion detection module, and would provide a basis for deciding what (if any) additional system or user parameters should be monitored.

Anomaly Detection

The goal of all anomaly detection modules is to identify the set of least frequent (lowest probability density) transactions such that the sum of their expected frequency (cumulative probability density) is less than some arbitrarily specified threshold (say 5%). In practice, the rules that would enable this identification need to be generated from a small sample of all possible transactions. Moreover, these rules need to be encoded in a succinct manner (to fit within the memory of modest computers) so that anomaly detection can be applied in real time. These constraints are by no means trivial. On the one hand, it is not atypical to generate anomaly detection rules on the basis of a sample that represents one one-millionth to one one-billionth of all possible transactions. On the other hand, this small (relatively) sample might include as many as several hundred thousand transactions (Vaccaro and Liepins, 1989).

A further complicating factor is that the majority of the variables are categorical, that is, their numerical values are arbitrary (such as port number) and any derived Euclidean distance is meaningless. Moreover, those variables that are continuous certainly do not satisfy the normality assumption of classical statistics. The significance of these observations in conjunction with the large quantities of historical data is that anomaly detection cannot be implemented by simply checking how frequently a transaction of interest has been seen in the historical data base nor by parametric statistical estimation techniques. Some sort of nonparametric density estimation approach is required.

W&S Approach

W&S solves the density estimation and mixed data (categorical data present with continuous data) problems by first clustering the continuous variables so that they can be treated as categorical. (For example, all the observations x satisfying the inequality 1.3 < x < 6.7 might be placed in the first cluster.) Next, based on the historical observations, a judiciously pruned forest of conditional rules is generated. Rules specify legal values (commonly observed values relative to the others) of "test fields" conditioned on the values in one or more of the other fields. For example, a rule might state that if the user is "gunar", the time is between 8:00 AM and 5:00 PM, and the day is Wednesday, then the legal ports are port1 and port2. (In this example, the test field is "port". As in this example, each rule has only one test field.) Consider a current transaction with "gunar" logged in Wednesday at 12:20 PM on port3. This transaction violates the previously stated rule, and therefore the rule contributes some evidence that the transaction is an anomaly. For any test field (subject to the pruning conditions and sufficient number of observations) rules are generated with all possible combinations of the other fields in the conditional side. Thus, rules will be formed that predict port on the basis of any combination of user, time-of-day, and day-of-week (individually or in combination); time-of-day on the basis of the other fields; and so forth. In this way, W&S can be thought to extrapolate the available information of what value combinations can be expected to be common and which are unusual: For each field individually, the corresponding tree of the W&S rule forest effectively partitions the space of possible transactions into complementary "rectangular" regions (of arbitrary dimension) that suggest evidence for or against the transaction being an anomaly (conditioned on the available information in the other fields).

The clustering algorithm and partitioning are illustrated in Figures 2. and 3. The clustering algorithm of Figure 2. produces separate categories whenever intervals of high density are separated by intervals of low density, and conversely. Figure 3. partially illustrates the partitioning of "anomaly regions" for time-of-day, conditioned on user and port jointly. Three groups of regions are illustrated: those conditioned jointly on "hank" and port1, those conditioned jointly on "gunar" and port1, and those conditioned on "gunar" and port2.

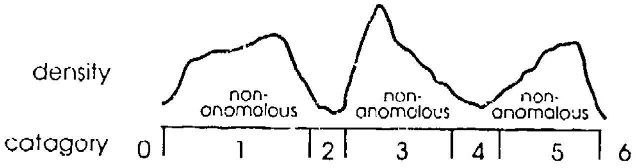


Figure 2. Illustration of W&S Clustering Algorithm

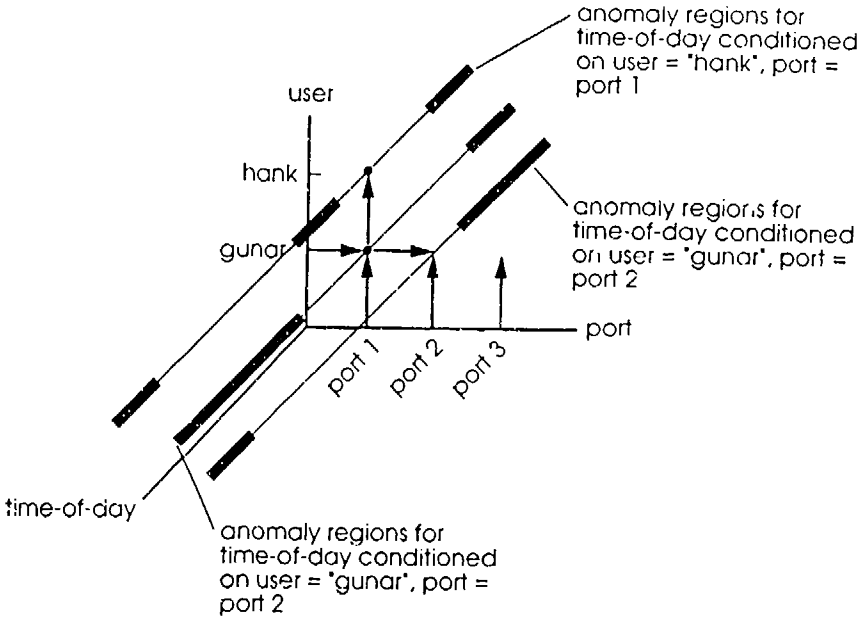

anomaly regions for time-of-day conditioned on user = 'hank', port = port 1
user
hank
gunar
anomaly regions for time-of-day conditioned on user = 'gunar', port = port 2
port
port 1
port 2
port 3
time-of-day
anomaly regions for time-of-day conditioned on user = 'gunar', port = port 2

Figure 3. Anomaly Regions for Time-of-Day Conditioned on User and Port

The justification for a multiplicity of rules (rule redundancy) specifying the value of any test field is that at the time of rule generation, it cannot be ascertained at what level of detail an incoming transaction might match the conditional side of a rule. Ideally, applicable rules should be maximally specific, but the historical data may not support equal levels of specificity in rules across all users, ports etc. Thus, the following three rules might be generated from the historical data:

rule1: if 10:30 AM - 1:45 PM; then either port1, port2, port3, or port4.

rule2: if "gunar", 10:30 AM - 1:45 PM; then either port1, port2, or port3.

rule3: if "gunar", 8:00 AM - 5:00 PM, Wednesday; then either port1 or port2.

A later transaction with user "gunar" at 12:20 PM on Thursday matches rules 1 and 2. A transaction with user "gunar" at 12:20 PM on Wednesday matches all three rules. A transaction with user "hank" at 12:20 PM on Wednesday matches only the first rule; hank probably did not have enough historical transactions on Wednesday to generate a counterpart to rule3. This is further addressed in the paragraph on rule pruning in the section on W&S details.

W&S Details

The approach as described to this point still leaves a number of important questions unanswered. How exactly is the rule forest grown and pruned? Once a rule forest is generated, a transaction under investigation will typically match a number of rules. Which of the rules should be considered in the determination of whether or not the transaction is an anomaly? Moreover, how much weight should be given to the various rules? Should passing rules be considered as well as failing rules? The rules in the rule forest address individual fields. How should evidence for or against a transaction be combined from the evidence about the individual fields? How can evidence about individual transactions be combined to help make decisions about user (logon) sessions or about the computer system as a whole. (For example, whether a distributed attack is being mounted)? What assistance might be provided to the security officer to aid in the interpretation of W&S output? These questions are briefly addressed in the remainder of this section.

W&S generates rules specifying legal values for each "test" field conditioned on all previously observed values in arbitrary combinations of the other fields (subject to the pruning rules). Legal values are specified by a set-wise complement: the historically observed values minus those of least frequency whose cumulative frequencies most nearly approach a given threshold. Thus, if a rule were being generated for port, conditioned on the user being "gunar" and the time-of-day being 8:00 AM to 5:00 PM, the following ports and corresponding frequencies might be observed in the historical data: port1, 60 observations; port2, 25 observations; port3, 11 observations; and port4, 4 observations. If the threshold were set at 0.05, the rule would specify: if gunar and 8:00 - 5:00, then either port1, port2, or port3.

Rule forest pruning is done in two stages. The first stage consists of stopping criteria which specify that a given rule be dropped and that the corresponding branch of the tree of the rule forest no longer be expanded. The second stage prunes out the "uninformative" rules left by the first stage. The first stage criteria are triggered by the following conditions: 1. too many legal values in the test field., 2. insufficiently many historical observations to support the rule, 3. an arbitrary depth cut-off, 4. conditioning on values previously determined to be anomalous (by rules earlier in the forest). Post pruning includes the pruning of descendent rules which specify the same legal values as their antecedents. For example, if two rules both specify legal values for the same test field, and rule1 is given a,b,c:f; and rule2 as a,b,c,d:f; then rule2 would be pruned (unless it had substantially greater strength -- see the next paragraph). Similarly, rules which do not constrain the legal values are pruned.

One of the most subtle issues in W&S is the determination of the strengths (weights) to be assigned to the rules. Ideally, the strengths should reflect the confidence that the rules flag transactions that should be flagged, and don't flag those that shouldn't. For example, consider the rule a,b:c. If this rule were based on 1000 historical transactions, one would have more confidence in it than if it were based on 20 transactions. Rules are assigned "passing" strengths and "failing" strengths. The passing strength is used in combining evidence if a transaction passes the rule; the failing strength is used upon rule failure. These strengths are determined as finite sample corrected maximal likelihood estimates (Howard, 1970). Thus, if N historical transactions matched the rule (the conditional --"if" -- side of the rule) and A of these transactions failed the rule, then the failing and passing strengths are determined to be proportional to (N+2)/(A+1) and (N+2)/((N-A+1), respectively. This strength assignment remains a focal point of continuing research. Currently, evidence from all matched rules is used in the determination of anomalies. This has some distinct disadvantages, the principle one being the multiple counting of dependent evidence. An alternative under investigation is the use of directed graphs. For example, for the test field f, a large number of rules potentially apply. These rules can be partially ordered in a directed graph as illustrated in Figure 4., below. (In this figure, the node " 0 " refers to the unconditioned rule for field f -- (0:f) -- , the node "a" refers to the rule for field f conditioned on a specified subset of values of field "a", and so forth.) A current transaction would be evaluated in terms of the "maximal elements" of the directed graph that it matches. Thus, if the transaction matches nodes 0, b, c, d, bc, bd, cd, bcd, e, de, and g (nodes corresponding to bcde, bcdeg .. are to have been pruned by the pruning rules in this hypothetical example), and fails all but g, then the evidence for and against field f would be computed in terms of the failing strengths at nodes bcd and de, and passing strength at node g. This directed graph approach to combining evidence helps assure that only "independent" evidence is combined.

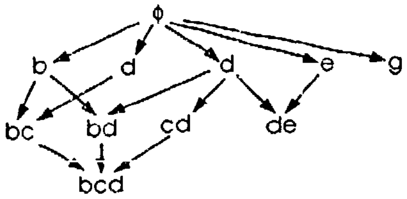


Figure 4. An Illustration of a Directed Graph (for Field f)

For each test field, a "figure of merit" (FOM) is calculated in terms of the standard error: Let E( ) and std-dev() be the expectation and standard deviation operators respectively. Then for a transaction T of interest and field f,

$$FOM(f,T) = (F - E(F))/std\text{-}dev(F)$$

where F is determined as the difference between the failing strengths of the (maximal) rules (of the f-field directed graph) failed by the transaction T and the passing strengths of the (maximal) rules passed.

Whether or not a transaction is flagged as an anomaly is determined in terms of the sum of the figures of merit for the various fields (perhaps weighted by the "importance" of the field). The "fields" of an anomalous record are those with the largest figures of merit.

The "incongruity" of a session is determined by the cumulative figures of merit associated with an uninterrupted (fixed) user-port combination. Overall system activity is monitored by the determination of figures of merit summed over system attributes such as ports or input-output activity.

Currently, the determination of the rules failed and passed, the computation of the figures of merit and the various measures of session incongruity are all computed within W&S. Under consideration is the separation of functionality: determination of raw evidence by W&S; processing of this evidence by a postprocessor. In this way, W&S could be tailored to have varying sensitivities towards different users and threats (based on the security officer's experience with the users and threats).

SUMMARY

The concept of an overall computer security system has been introduced the role of anomaly detection in such a system has been described. One approach to anomaly detection W&S has been reviewed. Although W&S cannot yet be described as a mature system, W&S's overall framework engenders considerable confidence that the module can be tuned to perform as desired. Nonetheless, the details of the actual implementation are continually being modified as experience is gained. For example, issues currently under review include the best formulation of the clustering algorithm and the most suitable functional form for the assignment of the rule strengths. On the other hand, even with some ambiguity regarding the best W&S configuration, performance to date has been encouraging. In every test to date, W&S has uncovered previously unknown inappropriate system activity and security shortcomings, such as for example, a process continuing after an electrical storm disrupted a remote connection. Efforts are continuing to integrate W&S into an overall security system and to further establish its soundness through rigorous analysis. To this effect, W&S is currently in beta test at LANL, Oak Ridge National Laboratory (ORNL), and other test sites.

REFERENCES

[1] Clyde, A. R., (1987). "Insider Threat Identification Systems," Proceedings of the 10th National Computer Security Conference, 343-356.

[2] Denning, D. E., (1987). "An Intrusion-Detection Model," IEEE Transactions of Software Engineering, vol SE-13, no. 2, 222- 232.

[3] Denning, D. E., D. E. Edwards, R. Jagannathan, T. F. Lunt, and P. D. Numan, (1987). A Prototype IDES: A Real-Time Intrusion-Detection Expert System.

[4] Hansen, J. V. and W. F. Messier, (1986). "A Knowledge-Based Expert System for Auditing Advanced Computer Systems," European Journal of Operational Research 26, 371-379.

[5] Howard, R. A., (1970). "Decision Analysis: Perspectives on Inference, Decision, and Experimentation," Proceedings of the IEEE, vol 58, No. 5, 632-643.

[6] Lunt, T. F., R. Jagannathan, R. Lee, S. Listgarten, D.L. Edwards, P. G. Neuman, H. S. Javitz, and A. Valdes, (1988). "IDES:Then Enhanced Prototype, SRI International," SRI-CSL-88-12.

[7] Lunt, T. F., (1988). "Automated Audit Trail Analysis and Intrusion Detection: A Survey," Proceedings of the 11th National Computer Security Conference, 65-73.

[8] Michalski, R. S., J. G. Carbonell, T M. Mitchell (eds.), (1983). Machine Learning, Tioga Publishing Company, Palo Alto, CA.

[9] Michalski, R. S., J. G. Carbonell, T M. Mitchell (eds.), (1986). Machine Learning, Volume II, Morgan Kaufmann Publishers.

[10] Quinlan, J. R., (1983). Learning Efficient Classification Procedures and their Application to Chess End Games, in Michalski, Carbonell, and Mitchell (eds.), Machine Learning, Tioga Publishing Co., Palo Alto, CA., 463-482.

[11] Sebring, M. M., E. W. Shellhouse, M. E. Hann, and R. A. Whitehurst, (1988). "Expert Systems in Intrusion Detection," proceedings of the 11th National Computer Security Conference, 74-81.

[12] Smaha, S., (1988). HAYSTACK: An Audit Trail Analysis System for Intrusion Detection, Tracor Applied Sciences, Inc., Austin, TX., personal communication.

[13] Tener, W. T., (1988). Discovery: An Expert System in the Commercial Data Security Environment, TRW Information Services Division, Orange, CA., personal communication.

[14] Vaccaro, H. S. and G. E. Liepins (1989). "Detection of Anomalous Computer Session Activity," IEEE Symposium on Research in Security and Privacy.

Computer Based Instruction for Computer Systems Security Officers - An Example by the Air Force Cryptologic Support Center Kelly AFB, San Antonio, Texas

BACKGROUND

This presentation will describe the Computer Based Instruction (CBI) effort done for the Air Force Cryptologic Support Center (AFCSC) at Kelly AFB, San Antonio, Texas. The need to reach Major Commands (MAJCOMs), Direct Reporting Units (DRMs), Special Operating Agencies (SOAs) and other Air Force personnel prompted AFCSC to explore a cost effective means to provide computer security awareness training.

Introduction

An increased need for computer security awareness training is a direct result of technological advances in automated systems. The wide use of computers in defense installations and particularly the Air Force requires the prudent application of security policies and procedures. The increasing connectivity of systems that are spread geographically has introduced new security complexities and issues that need to be addressed by a standard distributed training program.

Through the 1970's and 1980's computer assisted and managed instruction, which together equate to Computer Based Instruction (CBI), received increasing attention in Government, the military and industry. CBI has now become accepted as a viable training alternative offering reduced student and instructor time and resulting cost savings. With the availability of authoring languages and computer technology the training professional has many alternatives to select the alternative that best meets the training requirements.

Purpose

This presentation is for the purpose of: (1) describing the requirements analysis needed for effective CBI, (2) demonstrating the credibility of the approach by referencing an effort being implemented by the US Air Force Cryptologic Support Center. The paper describes a methodology consisting of requirement analysis, hardware/software environment, CBI benefits as applied to a particular military setting. The techniques examples, and learning experience from this should be useful to those who are planning the acquisition or who have recently implemented a CBI system. An actual demonstration of the AFCSC CBI Courseware will be accomplished.

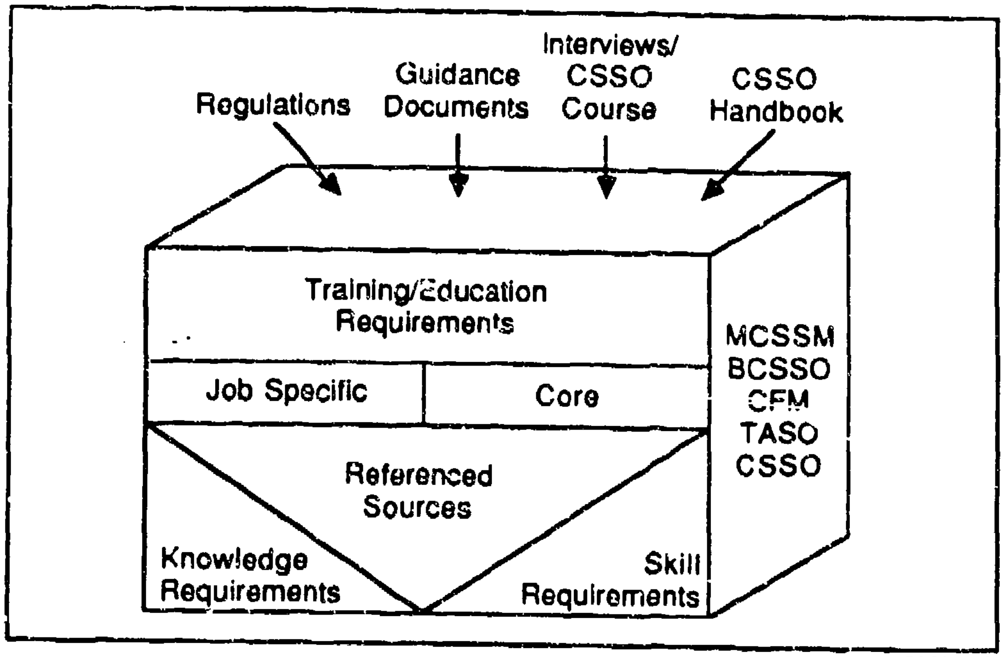
CBI DEVELOPMENT METHODOLOGY

A systematic approach was taken to identify the training and education requirements (TERs) and required skills and knowledge (S/Ks) for MAJCOM Computer System Security Managers (MCSSMs) Base Computer Systems Security Officers (BCSSMs), Computer Facility Managers (CFMs), Computer System Security Officers (CSSOs) and Terminal Area Security Officers (TASOs). The above positions represented the targeted audience as prescribed by Air Force Regulation (AFR) 205-16 entitled Communications Computer Systems Security Policy, Procedures and Responsibilities. Figure 1 depicts the process used to define the training education requirements from the reference material. The Instruction System development model utilized by the Air Force provided the foundation for the actual courseware development methodology. The ISD process is shown in Figure 2.

Information was gathered in the following sequence:

1. Identify appropriate computer security reference materials. Consult references for complete listing of documents.

2. Analyze the high level topics and identify appropriate subtopics for clarification of information content, noting the source of supporting information.



CS89-466

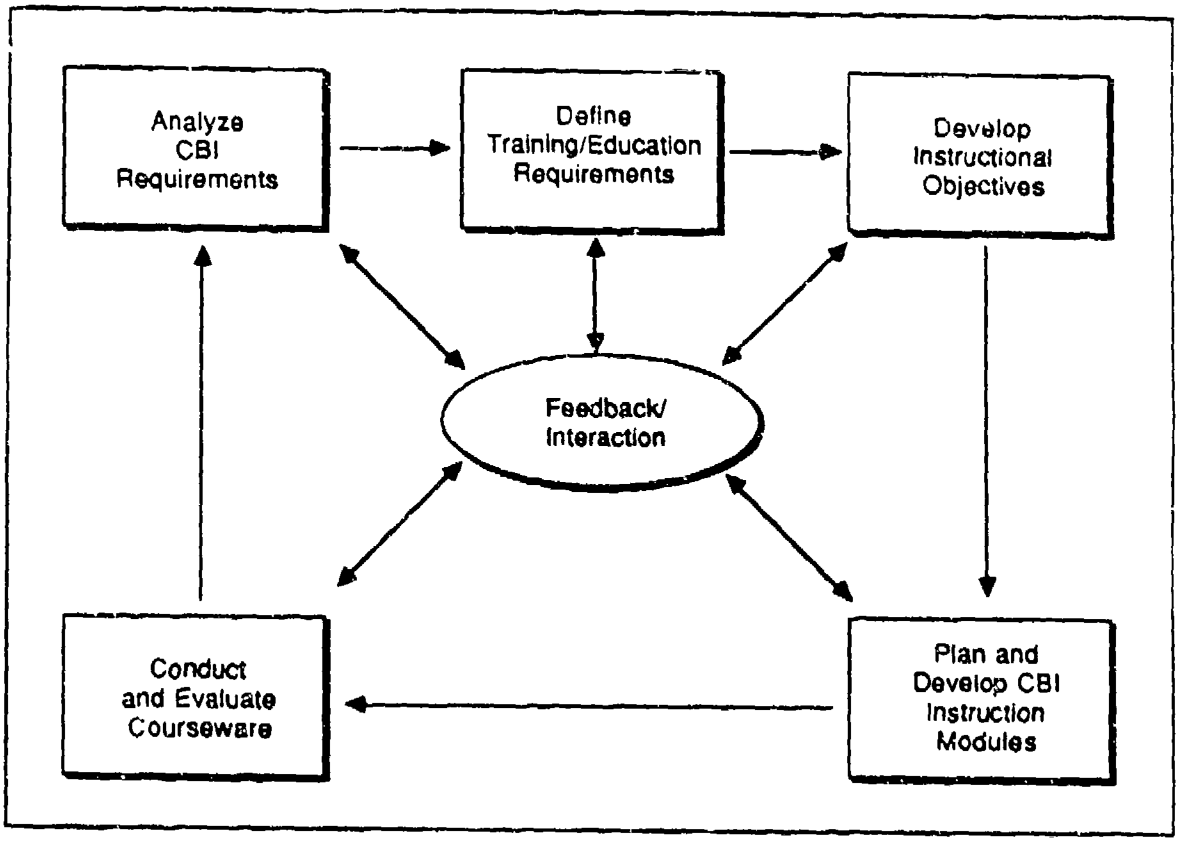
Figure 1. Requirement Identification

3. Analyze the high level topics and identify appropriate subtopics for clarification of information content, noting the source of supporting information.

4. Review high level topics and subtopics to determine if additional topics were required based on engineering experience, interviews with Air Force personnel and further analysis of reference materials.

5. Identify requisite S/K's for each identified TER. This information was derived from an analysis of the source information for each high level topic and associated subtopics.

6. Identify training requirements for each security officer as either Core Knowledge or Specific Training requirements. The criteria used for the core or specific training determinations were based on analysis of AFR 205-16, other reference materials, and personal interviews with Air Force personnel.


Analyze CBI Requirements
Define Training/Education Requirements
Develop Instructional Objectives
Feedback/ Interaction
Conduct and Evaluate Courseware
Plan and Develop CBI Instruction Modules

CS89-469

Figure 2. CBI ISD Methodology

7. Enter data into DBMS for manipulation and report generation. Databases were created containing the following:

   - o Training/Education Requirements (TER's)
   - o Information Source References (REF's)
   - o Skill/Knowledge Descriptions (S/K's)
   - o Cross reference files showing the relationships of TER's to S/K's, and TER's to REF's

MODULE IDENTIFICATION

Methodology Used To Identify Modules

The field of computer security is complex and ever changing. To develop the candidate CBI modules we explored two methods that offered a means to tap and pool the Project Teams judgment and expertise. The Delphi technique was used to prioritize and determine the logical sequencing of course material by topical area. Once the results of the Delphi were complete, the Nominal Group Technique (NGT) was utilized to finalize the logical sequencing of the course modules. The results of the NGT, as to module sequencing for CBI Courseware, are presented in Figure 3.

Module 01	Overview of Security
Module 02	Analysis of Security Architectures
Module 03	Security Requirements Analysis
Module 04	Overview of Certification/Accreditation
Module 05	Determining System Treats/Vulnerabilities and Countermeasures
Module 06	Details of Risk Management/Risk Analysis
Module 07	Security Incident Determination and Reporting
Module 08	Security Management of Communications Systems and Networks
Module 09	Management of Media in Security Environments
Module 10	Security in Day-to-Day Operations

Figure 3 - Order of Presentation and Development Modules

Once the module sequencing was established behavioral (enabling) objectives were established for each module. The objectives were based upon the training/education requirements and associated skill/knowledge requirements for module topics.

CBI Module Objectives

In Module 01, Overview of Security, the student will understand the reason for being concerned with security in automated environments and what the monetary costs of security violations can be. The student will also get a preview of the other CBI training modules that are available. The student will be able to describe the roles and responsibilities of the various positions involved in computer security, interpret computer security requirements, describe the requirements for planning and implementing a security program, describe the content and use of various security documents, and identify and prepare appropriate security training based upon requirements.

For Module 02, Analysis of System Architectures, the student will be able to describe security operations analysis, describe facility Risk Analysis/Certification Administration and the impacts of the environment. The student will also be able to describe computer facility requirements for processing classified/sensitive information for various system architectures, explain applicable security measures for various system architecture, interpret security impacts on small computer operation architectures, identify and use network/communication checklists and guides, and understand the risk analysis process for network architectures.

Module 03, Security Requirements Analysis, will enable the student to describe AFCSC services, interpret data derived from the architectual analysis of the sample system, understand the OMB, DoD, FIPS, AF and other security related publications for policy, procedures and guidelines. He/she will be able to interpret the implications of the evaluation criteria including: closed versus open environments, classes of systems, and use of tables to determine minimum user clearances and risk index, describe security modes of operation, and determine the applicable security mode for a sample software/hardware/data architecture.

In Module 04, Overview of Certification/Accreditation, the student will be able to identify and understand the use of security checklists and threat vulnerability guides, understand the requirements and components for preparing an accreditation package, understand the risk analysis process and documentation required for certification/accreditation of a system, prepare and execute a security test and evaluation plan, understand and determine residual risks of a system.

For Module 05, Determining System Treats/Vulnerabilities and Countermeasures, the student will be able to identify and understand the use of threat/vulnerability checklists and guides, identify and use a specific checklist/guideline for a particular security situation.

Module 06, Details of Risk Management/Risk Analysis, the student will be able to define and explain the risk management life cycle, explain the concepts, types of risks and the steps involved in completing a risk analysis, perform a risk assessment of a system, identify and perform the steps associated with a security test and evaluation, and describe and determine residual risks for a sample system.

In Module 07, Security Incident Determination and Reporting, the student will learn to describe rules and requirements for security monitoring, describe and understand the different methods of active versus passive monitoring, use the threat monitoring and severity/likelihood checklists, describe and understand methods for access control including: reasons to deny access, and file protection mechanisms and control, use access control mechanisms and file protection procedures, describe password management and protection requirements. The student will also be able to identify fraud, waste and abuse (FWA), and use methods to prevent FWA.

In Module 08, Security Management of Communications Systems and Networks, the student will be able describe the security requirements for networks, describe the security requirements for communications, describe and interpret the Network Evaluation Criteria, describe and use the methods for monitoring in a network environment, describe procedures of and conduct a CSESP IAW AFR 56-50, use network/communications threat/vulnerability guidance, describe data encryption/COMSEC requirements, describe types of cryptographic equipment and keying material. He/she will be able to describe and understand security requirements for LANs including protection methods, use of terminals on LANs, audit trails for LANs, access control of LANs, set LAN access attempts parameter, describe security requirements for long-haul/wide-area networks including: protection of dedicated lines, use of dial-up lines, audit logs, and network access control procedures, describe and know how to perform a network Risk Analysis, describe the procedures for networks used inter/intra commands/services/agencies, describe security requirements for using electronic mail systems and telephones including ones with displays, describe methods of protecting communication lines, describe types and understand use of PDS/PWDS and approved secure fiber optics, describe TEMPEST requirements to control emanations, apply procedures and interpret reports in TEMPEST required operations.

In Module 09, Management of Media in Security Environments, the student will be able to describe the magnetic media control requirements of CSC-STD005-85 and Guideline 06, identify the types of magnetic media to be controlled, describe the media labeling requirements of DoD 5200.1-R/AFR205-1 and other guidance, describe the proper methods to mark media, describe and understand the implications of the media storage requirements of AFR 205-16 Attachment 14, describe the criteria and requirements for backup/recovery of media, and describe the requirements of CSC-STD005-85 for declassification, degaussing, and destruction of media.

For Module 10, Security in Day-to-Day Operations, the student will be able to describe major computer operations security provisions of AFR 125-37, describe the use of configuration management in computer operations, use guidance to produce configuration control practices, describe sound safety practices, describe and understand the security implications of maintenance requirements, prepare a maintenance schedule, describe the requirements for transporting classified information, describe and understand the implications of the personnel security requirements of DoD 5200.2R/AFR 205-32, and know methods and events to look for in identifying potential system abusers.

The function of the CBI training material is to augment standard Air Force security training. The behavioral objectives for each module form the focus of the material presented.

PERFORMANCE REQUIREMENTS

Each of the ten (10) modules were designed with the following considerations:

a. Each module will require approximately thirty (30) to sixty (60) minutes to complete, assuming a minimum of remediation learning required.

b. Brevity - Are the displays simple and well-organized? Do the displays provide high information transfer?

c. Consistency - Are the displays consistent from one display to the next, thus developing user confidence in the module?

d. Flexibility - Does the module adapt to individual differences in background?

e. Compatibility - Does the module focus on the subject to be learned, or does the student focus on operating the learning system?

f. Responsiveness - Is immediate feedback provided to the student (except in the case of module tests where the student must complete the test before receiving his feedback)?

g. Does the computer managed instruction provide the trainer or supervisor with sufficient information so he can monitor the student's progress?

HARDWARE/SOFTWARE ENVIRONMENT

The CBI courseware is designed to operate in the USAF's standard Zenith AT compatible hardware environment. This basic hardware configuration consists of those items defined in Figure 4. Figure 5 presents the two types of Cathode Ray Tubes (CRT) or monitors that may be used.

Model No.	Description	Quantity
ZFX-248-50	Zenith AT compatible computer system	1
HE-150-192	360KB floppy disk drive system	2
HE-181-5188	80286 8MHZ CPU Card w/ 512KB RAM	1
HE-181-5187	Input/Output Card	1
Z-439	Enhanced Display Adapter	1
Z-304	Sync/Async Serial Card	1
HE-150-234	Floppy/Hard disk controller	1
HE-281-32	Z-200 Keyboard Assembly	1
OS-63-41	MS-DOS 3.2	1

Figure 4 - CBI Target Hardware Configuration

Model No.	Description	Quantity
ZVM-1380	RGB Color Monitor with EGA Board	1
ZMM-1470-G	Monochrome Monitor	1

Figure 5 - CBI Supported CRT Output Device

REQUIRED SOFTWARE CONFIGURATION

The CBI courseware consists of a set of floppy diskettes. One diskette contains the programs required to run the lessons and the remaining diskettes contain the introductory material and the individual courseware modules. Except for the need of the MS-DOS operating system, no other software is required. The design was on a dual floppy 360k system as the AF has over 14,000 CPU'c with this configuration. The courseware will also run on a single 360k floppy with a hard disk.

RESULTS

An actual demonstration will portray the CBI courseware that was described in this paper.

REFERENCES

Reference materials used in defining the training/education needs of Air Force security personnel include:

1. AFR 205-16, ADP Security Policy, Procedures and Responsibility (Draft), 28 August 1987.

2. DOD-STD-5200.28, DOD Trusted Computer System Evaluation Criteria (Orange Book), December 1985.

3. CSC-STD-002-85, DOD Password Management Guideline, 12 April 1985.

4. CSC-STD-003-85, Computer Security Requirements, Guidance for Applying the DOD Trusted Computer System Evaluation Criteria in Specific Environments, 25 June 1985.

5. CSC-STD-004-85, Technical Rationale Behind CSC-STD-003-85, Computer Security Requirements, 25 June 1985.

6. NCSC-WA-002-85, Personal Computer Security Considerations, December 1985.

7. ESD-TR-86-277, Risk Analysis Environments Guidelines, September 1986.

8. AFSEC Guideline 01F, Computer Security Incident, 27 January 1986.

9. AFSEC Guideline 02A, Security in Mission Critical Resources Acquisition, 1 February 1985.

10. AFSEC Guideline 04, Guidance in Performing a Risk Analysis, 1 March 1985.

11. AFSEC Guideline 08, Control and Prevention of Computer Abuse, 18 May 1981.

12. ATC-Student Text, Computer Systems Security, 3 April 1987.

13. ATC-Student Handout, Computer Systems Security ADP Security Guideline, 5 January 1987.

COMMUNICATIONS-COMPUTER SYSTEMS SECURITY VULNERABILITY REPORTING PROGRAM (CVRP)

CAPT LEE SUTTERFIELD
CAPT GREGORY B. WHITE
Networks and Computer Systems Security
AFCSC/SRE
Kelly AFB, TX 78243-5000
(512) 925-2386

1.0 INTRODUCTION

The Air Force Cryptologic Support Center (AFCSC) designed the Communications-Computer Systems Security Vulnerability Reporting Program (CVRP) to respond to several security problems facing the Air Force. The contents of this paper form the core of the official Concepts of Operations for the Air Force CVRP currently being implemented. The AFCSC, in its role as executive agent for COMPUSEC, COMSEC, and TEMPEST for the Air Force, has established the CVRP to focus limited security resources where they are needed most. Several aspects of the CVRP represent a departure from the primary emphasis of the past few years in each of the security disciplines. However, the CVRP will comply with all national policies and directives for COMSEC, COMPUSEC, and TEMPEST.

The CVRP is a combination of administrative controls, reporting procedures, specially developed software, research and development (R&D) efforts, and special survey and analysis capabilities designed to identify and develop countermeasures to known risks to Air Force communications-computer systems. It will provide a forum to identify and analyze system susceptibilities, security environment, and vulnerabilities. The CVRP will also establish a single office to identify and validate the threat to computer systems and report findings in a timely manner throughout the security chain of command. It will also direct the development and implementation of countermeasures to specific risks in the field.

Part of the basis of the CVRP is Department of Defense Instruction (DODI) 5215.2.. The Computer Security Technical Vulnerability Reporting Program (CSTVRP), as described in DODI 5215.2, requires Department of Defense (DOD) personnel to report all security vulnerabilities of DOD-owned computer systems. The CVRP will satisfy all of the reporting requirements of the CSTVRP. However, the CVRP is different from the CSTVRP in that it will address computer security (COMPUSEC), TEMPEST, and communications security (COMSEC) as an integrated effort.

In the past, the DOD computer security effort has concentrated on the development and deployment of the Trusted Computing Base (TCB) as described in DOD STD 5200.28. If the Air Force were able to field large numbers of TCBs soon, many of the more pressing computer security problems facing the Air Force would be solved. However, after many years of work through the National Computer Security Center, the government has stimulated industry to produce only a handful of TCBs of limited applicability.

Even if large numbers of TCBs were available tomorrow at a reasonable cost, the total cost to the Air Force to replace all operating systems would be prohibitive. While the TCB concept holds great promise, it will not meet the security needs of the Air Force for at least 7-10 years. In addition, a basic assumption of the CVRP is, even if TCBs are put in place, there will always be a need to report, analyze, and develop countermeasures for new vulnerabilities and incidents as they are discovered. In other words, if security features can be built, they can be broken.

The most important lesson learned about communications-computer security in recent months is, even though many of the issues are highly technical in nature, most of our security problems originate with people. First, education and awareness is all-important. We can avoid most communications-computer security incidents if system managers practice very simple security techniques. Second, we need door rattlers. No matter how good the education and awareness program, people will leave doors unlocked, physical, digital, and otherwise. We must have the ability to walk the digital hallways and close the doors when needed. The CVRP will provide a formal, organized process for doing that. Third, and by no means least, we must collect the information necessary to help plan the distribution of our limited communications-computer security resources for long term security.

The CVRP is a new approach to securing Air Force computer systems. AFCSC developed the CVRP as a direct result of recent lessons learned from the growing volume of computer security incidents along with the need to integrate the disciplines of COMPUSEC, TEMPEST, and COMSEC.

2.0 THREAT AND SECURITY ENVIRONMENT PERSPECTIVE

Each aspect of the CVRP is the result of practical lessons learned from the analysis of recent communications-computer security incidents as well as an evaluation of communications-computer and network technologies. A list of the most important of those lessons and background on each is described below.

2.1 Sensitive Unclassified Systems

The Department of Defense must now secure sensitive unclassified systems. Until recently, we directed the majority of the computer security program toward securing only classified systems, and much of that effort was expended on the development of TCB technology. However, in recent years it has become clear that we can no longer ignore the large number of unclassified systems processing highly sensitive information.

In addition, there are several categories of sensitive unclassified systems, each of which has unique security requirements. For example, life-critical systems may take priority over mission critical systems depending on the circumstances. We must also provide special protection for Privacy Act information. All of these special requirements makes security decisions related to these systems more complicated.

The National Telecommunications And Information System Security Committee (NTISSC) published NTISSP No. 200, 15 Jul 87, stating all government systems, including those systems processing sensitive unclassified information, must provide a C2 level of trust as described in the DOD STD 5200.28 by 1992. There is currently a limited number of evaluated TCBs and a long lead time is necessary to develop and successfully field a TCB. In addition, the Air Force has limited resources available to change a large portion of existing Air Force computer operating systems to TCBs.

The lead time for fielding large numbers of TCBs with higher levels of trust (i.e., B1, B2, etc.) will be at least as long as for C2 systems. Therefore, we must concentrate on providing equivalent C2 capabilities outside of the TCB via procedures, controls, and other security practices. In addition, recent experience clearly shows that TCBs are often installed and used incorrectly. Installing a TCB is no guarantee of security. We must still deal with a constantly changing environment and technology.

The requirement to secure all Air Force computer systems, while desirable, is not possible if TCBs are to make up the core of the computer security program. Recent security incidents have shown that the security of existing computer systems, most of which are not trusted, can be greatly enhanced by using the security features that are already available on current systems. The following section will describe in some detail the findings of several case studies involving the security of Air Force systems. It will also provide some perspective on the weaknesses in the Air Force computer security program that the CVRP will address.

2.2 Connectivity of Computers and Networks

The growing connectivity of networks and computers has clearly changed the nature of the threat posed to Air Force systems. The Air Force is connecting computers that process information about operations, logistics, personnel, administration, finance, medical, and other subjects vital to daily Air Force business in ever-increasing numbers via local and wide area networks. Recent experience suggests that the connections between computers are increasing so fast that users and system managers are not always sure of exactly who can obtain electrical connection to their system or files. Often a particular computer system may be accessible through a second, third, or fourth level of connectivity that isn't apparent to the owner of that system. Couple this problem with lax security discipline, and the environment is ripe for serious exploitation of systems.

For example, personnel at the Lawrence Berkeley Labs (LBL), Berkeley, California, monitored a systematic attack on 450 computers connected to the Defense Data Network between late 1986 and late 1987. The intruder obtained some degree of access to over 60 of the computers he attacked through MILNET. He was able to obtain access to a programming environment on about 18 of the systems, and he gained full system manager privileges on 9 of the systems attempted. Only 2 system managers out of the 450 systems attacked are known to have detected the attack themselves.

2.3 Simple Hacking Techniques

All of the attacks in the LBL hacker case were conducted using very simple hacking techniques. This observation can't be over-emphasized. The hackers gained access through security holes that should have and could have been closed by system managers. Often, once the attacker obtained simple access to a system, he would seek information on who was currently logged on to the system. He would later use this list of valid users of the system to log in with their valid user names and try to guess passwords. About 36 of the attacker's attempts were successful in obtaining such user information. He usually tried only 4 guesses per computer and successfully penetrated 60 of the computers attacked.

2.4 Poor User Discipline

Once the LBL Hacker penetrated a system, he would search the users' personal files for information about other computers to which the users had authorized access. Unfortunately, many users kept files listing up to 20 other computer systems or networks that they regularly accessed. These files contained telephone numbers, system IDs, user IDs, and passwords. Although most system managers regularly remind users not to keep such information in their computer files, the attacker was able to gain access to many systems by exploiting this classic security error.

2.5 Poor System Manager Discipline

Computer system managers are the first line of defense for the security of Air Force computer systems. However, they are often poorly trained in the use of existing security features of an operating system. In addition, they are usually under pressure from users to minimize the impact of security on the operability of the system. Most external break-ins to computer systems happen because hackers exploit security holes in operating systems that can be closed by motivated and well-trained system managers.

For example, the intruder in the LBL case often accessed systems by entering standard user names looking for open maintenance accounts. For example, UNIX systems often have accounts for <guest>, <ingres>, and <uucp>, while VMS computers usually have <system>, <user>, and <systest>. System managers should close such accounts when no one is using them.

2.6 Operating System Vulnerabilities

Hackers can quickly exploit weaknesses in operating systems because of the increased connectivity of Air Force systems through both military and commercial networks. The following incident is described as an example of the potential danger in not quickly correcting operating system vulnerabilities in today's high-connectivity environment.

In 1988, a group of hackers exploited a technical vulnerability in a commercial operating system via worldwide networks. They used the Space Physics Analysis Network (SPAN) and the High Energy Physics Network (HEPNET) to attack a large number of Digital Equipment Corporation computers running version 4.4 of the VMS operating system. The flaw in the system allowed a user to gain access to the file that controls the system user access and privilege data. Once the intruders gained access to that file, they gave themselves full system manager privileges. This group was able to plant a sophisticated Trojan Horse program within the operating systems of over 100 computers. The connectivity allowed them to exploit a simple but devastating operating system vulnerability on a worldwide scale within months of the discovery of the vulnerability. Clearly, we must be able to determine countermeasures for vulnerabilities quickly in today's networked environment.

2.7 Security Monitoring

During the 12 months of attacks in the LBL case, less than 3 percent of the attacked sites noticed any attempts at unauthorized access. Although it's easy to watch for network log-in attempts, only a few system managers do so. Few managers look for warnings of problems and even fewer act upon these warnings. However, if Air Force system managers were trained in what to look for and how to respond, the security of Air Force systems would increase dramatically. This has been proven at several locations in the Air Force and within other government agencies.

2.8 Security Incident Procedures

Part of the CVRP will address the need for centralized threat and vulnerability assessment. It will also provide clear guidance for system managers in the field about countermeasures for security incidents. For example, another lesson learned from recent incidents relates to the way a system manager should handle a break-in attempt. At one point during the LBL investigation, the system manager at LBL notified the system manager of a particular Air Force site that the intruder had accessed the Air Force system and had obtained full system manager privileges. LBL explained that they had been tracking this hacker for 10 months and asked that the system manager at the Air Force site not alert the intruder. LBL asked the site system manager to shut down the system gracefully, perhaps using routine maintenance as an excuse, and assess the situation.

Unfortunately, the Air Force system manager took a parochial view of the situation and simply closed the hackers' accounts. It was clear from his actions that the system manager had detected the intruder. Fortunately, the hacker did not realize that LBL was monitoring his actions. If he had, the LBL investigation would probably have stopped and 10 months worth of work lost.

Although the Air Force system manager was following existing guidelines, he almost closed the most revealing case of potential computer espionage ever documented. He, like all other Air Force rystem managers, hasn't been given clear instructions regarding such situations because we have yet to form a clear policy for such incidents. The CVRP will provide that policy.

2.9 Tracing of Illegal On-Line Activities

The last major lesson from these recent security incidents highlights the need for a point of centralized coordination to trace unauthorized queries of Air Force computer systems back to their point of origin. Through the CVRP, the Air Force can use the same technology hostile groups use to illegally access and damage systems and catch them in the act. The LBL investigation took over a year to complete. The need to coordinate with up to ten organizations and agencies in several countries, plus the lack of clear policy regarding illegal access to on-line systems, caused unneeded delays. In this environment the hacker has the advantage. He is working on his turf against one poorly trained system manager at a time. If the current situation continues, he will become increasingly successful. The goal of the CVRP is to organize all communications-computer security resources available and direct them where needed and when needed.

3.0 CVRP BACKGROUND

Each function of the CVRP is designed to meet at least one of several constraints. First, the CVRP is a "threat-driven" program. Each major action through the CVRP will be directed at existing vulnerabilities for which there is a "validated threat" on record. The next section outlines this requirement. Second, the CVRP will provide for the DOD level reporting requirements mandated by the CSTVRP. Third, the CVRP will integrate the reporting and analysis functions of COMPUSEC, TEMPEST, and COMSEC.

The primary emphasis of the CVRP in the early stages will be on COMPUSEC since our greatest vulnerabilities exist in this discipline. Although TEMPEST and COMSEC are integral parts of the CVRP, each discipline has long standing procedures for handling the analysis and reporting of vulnerabilities that will not change overnight. These two disciplines will be phased into the formal reporting requirements of the CVRP as soon as possible. However, the performance of countermeasure assessments, electronic security surveys, and other functions of the CVRP will address all three issues as appropriate.

3.1 Threat-Driven Program

Implementation of the CVRP will require coordination at all levels within the Air Force and with several agencies and organizations outside the Air Force. The CVRP will facilitate the deployment of countermeasures in three different disciplines. All three disciplines have developed separately over the years and they each use key words and phrases in very different ways. In addition, the CVRP will deal with the intelligence community on a regular basis. This multi-disciplinary nature of the CVRP caused considerable confusion in the early days of the integration of communications-computer security. The CVRP will use a set of definitions that will satisfy all three security disciplines and the intelligence community.

3.1.1 Definitions. The most important definitions are referenced here instead of in a glossary because of the need for a clear understanding of certain concepts that are fundamental to the CVRP. These definitions were coordinated with several organizations outside the Air Force to "test the semantic waters" and adopted for the CVRP.

Threat--The potential for the exploitation of an existing vulnerability by a hostile entity.

Service Interruption Hazard--The chance that an action may occur which would have a detrimental effect on the operational integrity of a system.

Susceptibility--The lack of the ability of a system to prevent: 1) an electronic compromise of National Security Information or, 2) detrimental effects on its operational integrity.

Security Environment--Environmental security factors, in a particular installation, which could allow a system's susceptibility to be exploited and/or deactivated.

Vulnerability--A product of susceptibility and the security environment. A measure of the possibility of a successful exploitation.

Risk--A product of validated threat and vulnerability. A measure of the likelihood of successful exploitation of a system.

Countermeasures--Adjustments made to system susceptibility and/or the security environment which reduce the system vulnerability to a level which, with the threat, equals an acceptable risk.

3.1.2 CVRP Risk Model. The CVRP risk model will be used to develop and deploy countermeasures to confirmed risks to Air Force systems. Using the definitions given above and Figure 1.0, an explanation of the CVRP risk model follows.

In the past, most decisions about the deployment of countermeasures for communications-computer security problems were made based on identified susceptibilities of systems. For example, in the TEMPEST arena, we used considerable resources to minimize all compromising emanations even if the exploitation of those emanations wasn't likely. This led to wasted security dollars. The CVRP risk assessment model should help avoid similar waste in the future in all three security disciplines. Use of this model will be modified as necessary to comply with specific policy requirements. However, adhering to the primary CVRP goal will guide our actions.

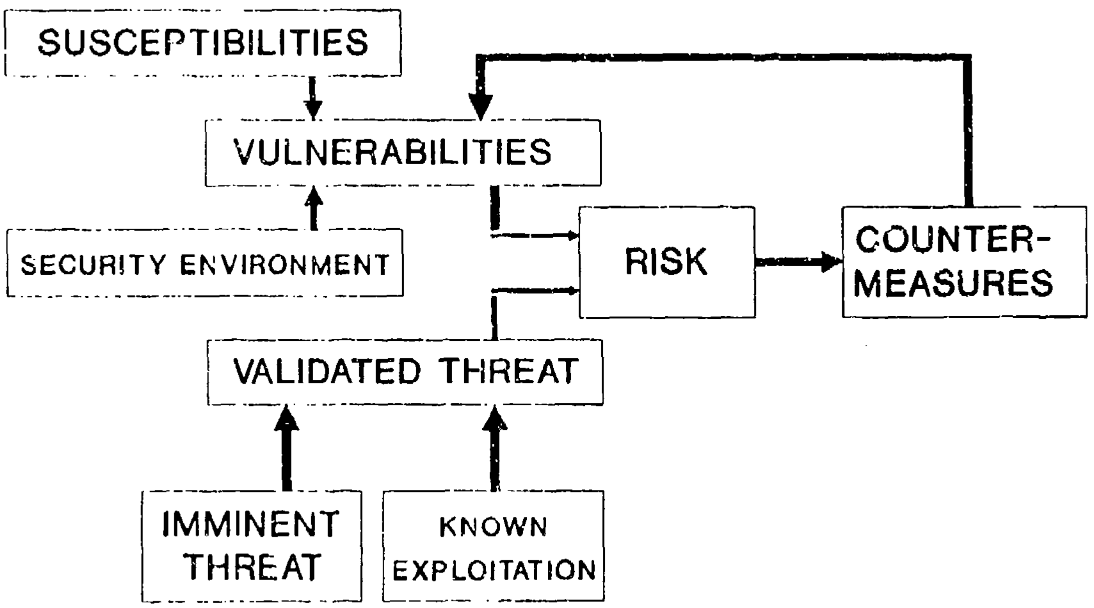
The goal of the CVRP is to field countermeasures based on clearly identified risks to specific systems. A risk must have the following three parts.

First, there must be a confirmed vulnerability of the system in question. A vulnerability is a system susceptibility that can be exploited because of the security environment in which the system is used. For example, suppose a given computer system radiates compromising emanations up to 20 feet, but the security environment is such that 100 feet of controlled space is available. A vulnerability does not exist because the physical access needed for exploitation is controlled.

Second, the sensitivity of the information will determine the level of risk. The Air Force must protect classified information and little risk can be accepted. We must also protect information subject to the Privacy Act and information that is critical to the operational security of the Air Force.

Third, there must exist a validated threat to a system before risk exists. In the past, the development of security countermeasures for communications-computer systems did not require validated threat position. A basic premise of the CVRP is that the Air Force must use its limited security resources where the exploitation of a given vulnerability is most likely. A validated threat position produced at the Air Force level and at the Defense Intelligence Agency will justify each major expenditure for communications-computer system security countermeasures.

THREAT-DRIVEN PROGRAM



CVRP RISK MODEL
FIGURE 1.0

There are two types of validated threat to Air Force communications-computer systems. The first type of threat is any known exploitation of existing Air Force systems by hostile organizations or persons, often referred to as the "smoking gun". The second type of threat is "imminent threat". Imminent threat will exist if a hostile organization has the technology, the organizational assets, and the demonstrated intent to exploit a known vulnerability. If a countermeasure has been effective it should significantly reduce the vulnerability and therefore the risk to an acceptable level.

3.2 CSTVRP

As mentioned earlier, DODI 5215.2 established the Computer Security Technical Vulnerabilities Reporting Program (CSTVRP) under the direction of ASD (C3I) as a means for reporting all demonstrable and repeatable technical vulnerabilities of computer systems. The CSTVRP provides for the collection, consolidation, analysis, reporting or notification of generic technical vulnerabilities, and dissemination of corrective measures.

The program focuses on hardware, firmware, and software weaknesses and design deficiencies in commercial products acquired by the DOD and those altered computer system products supporting standard military applications. Air Force participation in this program is expanded to include products developed by Air Force, other DOD, or private sources used on Air Force standard communications-computer systems. Responsibility for correcting vulnerabilities in commercial products will be assigned by a national level agency, usually to the owning vendor. The Air Force Cryptologic Support Center is responsible for ensuring vulnerabilities in Air Force standard products included under the expanded Air Force reporting program are corrected.

3.3 CVRP Mission Objectives

The Communications-Computer Systems Security Vulnerability Reporting Program (CVRP) includes all of the requirements for the CSTVRP plus the capability to handle hacking incidents, virus incidents, technical vulnerability reporting, and security surveys, and to organize the development of specific countermeasures for standard and embedded systems. Administratively, the CVRP will meet all the reporting requirements of the CSTVRP but AFCSC will provide specially developed software to facilitate participation of Air Force Computer Systems Security Officers (CSSOs) in the program. Details of the operational and functional requirements of the CVRP and its software will be described in Section 4.0.

The CVRP will direct its limited communications-computer security resources to prevent the exploitation of the most critical Air Force communications-computer resources where the greatest risk exists. The CVRP will provide a forum for the identification and analysis of system susceptibilities, security environment, and vulnerabilities; identify and validate the threat to computer systems; perform security surveys of organizations and networks; report those findings in a timely manner throughout the security chain of command; and facilitate the development and implementation of countermeasures to specific risks in the field.

4.0 THE CVRP PROCESS

4.1 Overview

The CVRP process will involve the collection and analysis of three types of information. The details of the methods of collecting these data will be outlined in paragraph 4.2.

First, AFCSC will collect sufficient data on each accredited Air Force computer system to identify that system, the key personnel responsible for the security for that system, and pertinent technical and environmental information. AFCSC will use this data to handle security incidents that may have Air Force wide implications and to conduct analysis of the overall computer security posture of Air Force organizations and systems. Computer Systems Security Officers (CSSOs) will forward this data to AFCSC through the MAJCOM Computer Systems Security Manager (MCSSM).

Second, AFCSC will collect vulnerability information and maintain a vulnerability database. This vulnerability data base will also satisfy vulnerability reporting requirements mandated by DODI 5215.2 for the CSTVRP.

Third, threat information, as defined in the earlier definitions section, will be collected and analyzed. This information will be all-source intelligence validated at the Air Force level as a minimum.

The three types of information mentioned above will make up the CVRP Database. AFCSC will use the CVRP Database to conduct Security Posture Assessments (SPAs), Vulnerability Report (VR) analyses, and countermeasure development. The focal point for all aspects of the CVRP is AFCSC/SRV.

4.2. Data Collection for CVRP Data Base

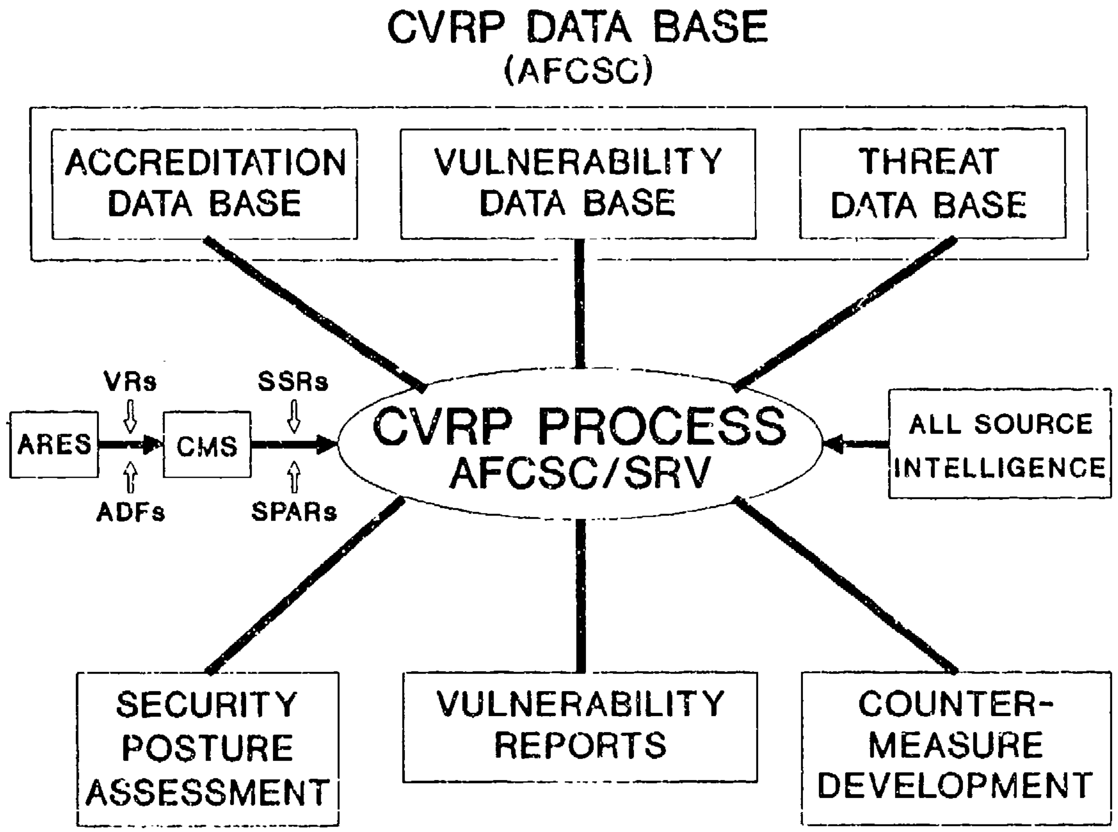
The collection of accreditation and vulnerability data starts with the performance of the risk analysis by the CSSO as shown in Figure 2.0. The Automated Risk Evaluation System (ARES) is the primary risk analysis tool for Air Force use. The system security officer performing the risk analysis will forward selected accreditation data up to the MAJCOM/SOA/DRU CSSM via the Communications-Computer Systems Security Management System (CMS). The CMS is a software tool designed to help the MCSSM manage the MAJCOM communications-computer security program. The CMS will also manage the collection of vulnerability information for the Air Force Vulnerability Data Base (AFVDB). (Paragraph 4.2.2 describes the CMS in more detail.) The CMS will also manage the collection of accreditation information which will be kept in the Air Force Accreditation Data Base (AFADB). The Air Force Threat Data Base (AFTDB) will contain validated threat information from all-source intelligence resources. These three data bases make up the CVRP Data Base maintained by AFCSC. Subsets of portions of these data will reside in the CMS program at the MAJCOM/SOA/DRU CSSM offices. A more detailed description of the above items follows.

4.2.1 Automated Risk Evaluation System (ARES). The backbone of the Air Force Communications-Computer Systems Security Program is the requirement for the completion of a formal risk analysis. AFR 205-16 requires that the user of a computer system perform a risk analysis before the Designated Approving Authority (DAA) can approve the operation of a particular computer system. The number of man-hours expended to satisfy this requirement Air Force-wide is considerable. Yet none of the resultant data generated during this process is available to plan the development of needed countermeasures and to assess the overall security posture of Air Force communications-computer systems resources.

ARES is an automated tool designed to make the risk analysis much easier for the average user to perform. In addition, ARES will provide additional functionality needed by the field user to maintain the overall security posture of the computer system at the highest possible level. ARES will describe to the user in detail what his responsibilities are in support of the CVRP, such as the reporting of technical vulnerabilities and hacking incidents. It will also provide a convenient tool to maintain an automated list of systems and authorized software for each system as part of the control for personal computer environments.

The ARES program will generate a number of reports needed for the accreditation process. It will also generate the Accreditation Data File (ADF) to be forwarded to the MAJCOM for the Accreditation Data Base (ADB). The MAJCOM will periodically forward an ADB update to AFCSC to update the AFADB.

When the risk analysis is done, the CSSO will download the ADF to a separate disk and forward that data to the MCSSM. The MCSSM will use the CMS software to manage the process. The MCSSM will forward the updates to the



CVRP DATA COLLECTION
FIGURE 2.0

ADB once each quarter to AFCSC. AFCSC will then load the data into the AFADB.

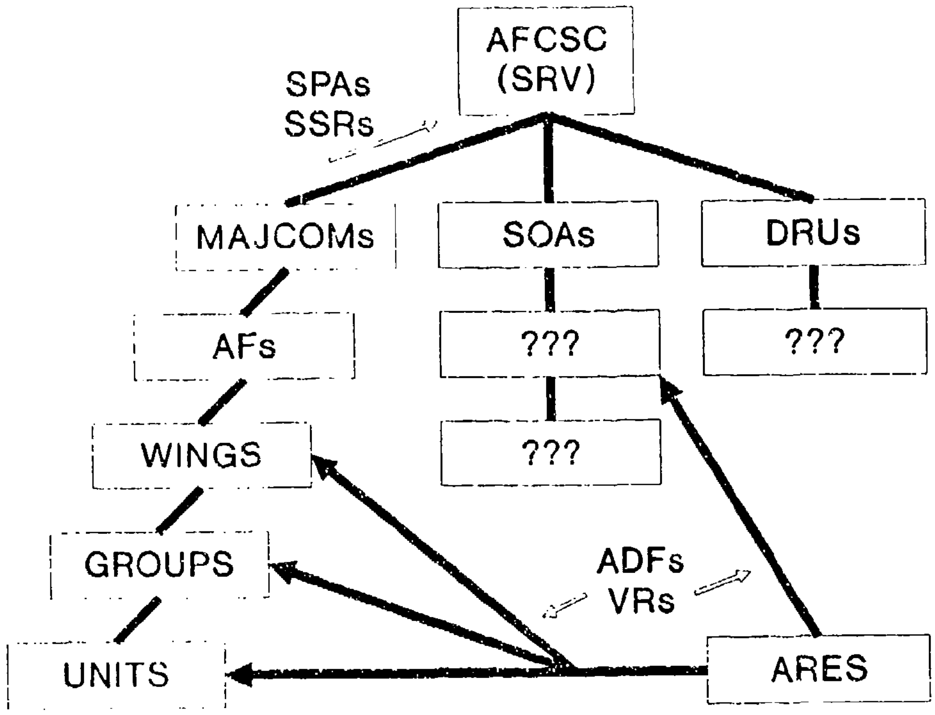
The data in the ADF provided by ARES will consist of the following fields as a minimum. The ADF will contain all of the information necessary to identify each computer system, its software, its peripherals, and the security environment that a given system security officer is responsible for. This information is consolidated by the ARES program during the risk analysis phase of system accreditation.

The file has two parts. Part 1 contains all the information needed to identify the organization and persons responsible for systems covered by a single risk analysis effort. Part 2 contains all the information needed to identify each computer system and all its components.

4.2.2 Communications-Computer Systems Security Management System (CMS). The CMS is designed to support MAJCOM/SOA/DRU CSSMs in performance of CVRP-related duties as well as other Communications-Computer Systems management functions. The CMS provides support to collect and organize accreditation and vulnerability data (two of the inputs to the CVRP process). It will also produce two of the CVRP outputs--Security Posture Assessment Reports (SPARs) and Vulnerability Reports (VRs) for the owning MAJCOM.

AFCSC will issue the CMS software to MAJCOM/SOA/DRU CSSMs (see Figure 3.0). These offices can then issue copies of CMS to subordinate units

CMS DISTRIBUTION



CMS DISTRIBUTION
FIGURE 3.0

within their organization who will act as a point of data consolidation for the accreditation data and vulnerability data being forwarded to AFCSC. The security classification guide for the CVRP will provide clear guidance on the classification of the CMS and other portions of the CVRP.

4.2.3 Validated Threat Information. AFCSC will direct all-source intelligence resources in the collection and production requirements for gathering all-source intelligence data. They will assist certain organizations in the Scientific and Technical Intelligence Analysis function. They will also build and maintain the AFTDB in support of the Communications-Computer Systems Security Program.

4.3 Security Posture Assessments

AFCSC/SR will also produce periodic and special SPAs as part of the CVRP. An SPA will be a report that provides perspective on the direction and status of the security posture of Air Force communications-computer systems. For example, they should be able to use the AFADB to determine the number of systems of a particular hardware/software suite that process classified information in the system high mode. AFCSC will use this information to evaluate the potential impact of a given policy decision on overall Air Force security and resources. As another example, specialized SPAs may describe the number of computers of a certain type and the connectivity of those systems. It will also include data from the AFTDB and the AFVDB as needed. Such an SPA would provide perspective on the scope and scale of effort needed to impact the security posture of those systems Air Force-wide.

The primary customers of SPAs will be Air Staff, MAJCOMs, SOAs, DRUs, and AFCSC itself. Each MAJCOM will be able to produce SPAs based on its own accreditation data base for their local DAAs when needed. AFCSC will assist the MAJCOM with these efforts when access to the whole CVRP Data Base is needed.

4.3.1 Electronic Security Survey Teams. The overall security posture of Air Force communications-computer systems is only as strong as the weakest link. Because of the extensive connectivity between DOD computer systems, the more weak links we leave, the more hidden "back doors" will exist. If we spend too many resources on one organization, then we will have less resources to spend on others. As we close all vulnerabilities in the first organization we will leave the "front doors" open at several other organizations.

The Electronic Security Survey Teams (ESSTs) will function as the primary source of on-site vulnerability analysis and security posture evaluation for all three security disciplines. They will concentrate on first level vulnerabilities caused by technical vulnerabilities of the hardware or software, poor user discipline, poor system manager discipline, inadequate administrative procedures, physical security, system connectivity, or technical security susceptibilities.

The primary mission of the ESSTs will be to test and assess the overall electronic security posture of an organization; they will concentrate on breadth of activity, not depth. They will look for first level vulnerabilities that are most likely to be exploited or lead to security incidents, not an in-depth analysis of all technical vulnerabilities. The teams will concentrate on performing security surveys at as many locations as possible.

The ESSTs will be requested by a unit commander through their MAJCOM CSSM via the submission of a Security Survey Request (SSR). The SSR will be reviewed by AFCSC and assigned a priority based upon the results of an SPA. All requests for surveys will be forwarded to the ESSTs who will schedule the surveys according to resource availability and the priorities assigned by AFCSC.

4.3.2 Special Test and Evaluation Capabilities. AFCSC will provide several forms of special test and evaluation capabilities. These resources will exist in the Product Evaluation Resource Center (PERC), TEMPEST Test Chambers, Advanced Techniques Lab, Prototype Lab, and the TEMPEST Test Teams. All of the resources above will be used to conduct in-depth analysis of technical vulnerabilities. The functionality of each may evolve over time to provide an integrated service to the field (that is COMSEC, COMPUSEC, and TEMPEST).

4.4 Vulnerability Reporting and Incident Handling

The second major function of the CVRP is Vulnerability Reporting. This function of the CVRP will require considerable technical resources. Some of those resources will be available within AFCSC and some will be available from other organizations such as Army, Navy, Department of Energy, Computer

Emergency Reaction Teams, National Institute for Standards and Technology and others.

AFCSC will perform two types of actions under the heading of Vulnerability Reporting. First, AFCSC will act as both the clearing house and central repository for all technical vulnerabilities of Air Force communications-compute systems. Second, AFCSC will act as the central point of coordination for the handling of communications-computer systems security incidents to include hacking incidents and virus outbreaks.

4.4.1 Vulnerability Reports. During a risk analysis or normal operation of a system, the user may identify a technical vulnerability in the system. Under the CVRP he is required to submit a formal report of that vulnerability to his MCSSM. The ARES program will provide the format of the Vulnerability Report (VR). The user will write the VR and submit it through the channels designated by the MCSSM. The MCSSM will conduct an initial analysis of the vulnerability and forward the VR with their perspective to AFCSC.

AFCSC will conduct a detailed analysis of the vulnerability and forward the final draft of the VR to a national level agency. An assessment of the technical validity of the report is essential. AFCSC will delegate that responsibility according to the nature of the vulnerability described. As part of that validation process, AFCSC will conduct a Countermeasure Assessment (CA) and recommend specific countermeasures for the vulnerability. The CA will take into account all aspects of risk as described above in the CVRP risk analysis model. All of the CVRP data bases will be used, the AFADB, AFTDB, and AFVDB.

CAs will begin with an analysis of the susceptibilities of the system in question. This will require a detailed look at the hardware/software to determine the technical requirements necessary for an effective countermeasure. Next, to determine if the vulnerability has applications throughout the Air Force, AFCSC will examine the environment in which the system operates. This may require an SPA to provide that perspective. The exact nature of the validated threat to the system will then be determined. Action will be taken only if exploitation of the vulnerability seems likely. Countermeasures will be developed as resources permit. The Air Force Consolidated Communications-Computer Systems Security Research and Development Program, headed by AFCSC, will initiate an R&D effort if appropriate. This process will be the same for COMPUSEC, TEMPEST, or COMSEC vulnerabilities.

Anyone at any level can initiate a vulnerability report, but AFCSC will make the final Air Force verification of the vulnerability and issue the final VR to a national level agency. Again, the threat to the system must also be sufficient to warrant an extensive use of resources.

4.4.2 Security Incident Handling. AFCSC will perform special "on-line surveys" in support of SPAs. As an example, suppose evidence is received that a particular hostile organization is conducting illegal on-line activities and exploiting a known vulnerability that exists in a commercial operating system used in a number of Air Force computers. Also suppose that the SPA suggests that a large percentage of these systems are processing sensitive unclassified information and are connected to wide area networks such as the Defense Data Network (DDN).

A small team could access the network and attempt to exploit each Air Force system using the same techniques used by the hostile organization. If a system is penetrated, the team could notify the system manager about the vulnerabilities so he could take the appropriate protective actions immediately. At the same time, statistics of success for each attempt and details of the successful techniques would provide invaluable security lessons for everyone involved.

At this point it would also be possible to leave a select few computer systems vulnerable and place special monitor systems on-line to alert system managers when an attack hits their system. Once the intruder attempts to attack a monitored system, the system manager would notify AFCSC who would centrally coordinate the tracing of that attack back to the point of origin to identify the culprit.

AFCSC will establish the necessary points of contact with the appropriate agencies for this activity. Memorandums of Agreement or Understanding will be established with these organizations as necessary.

4.5 Countermeasure Development

AFCSC is the primary Air Force point of contact for the development of countermeasures. It will have access to the all-source threat data that will be necessary to validate that the vulnerability does give rise to a serious risk. If the VR is verified as serious and countermeasures are needed, then AFCSC will request the development of appropriate countermeasures. The countermeasures may range from policy changes to new education and awareness efforts to the development of new hardware or software solutions. If the most effective countermeasures will require long-term efforts, then AFCSC will develop interim countermeasures as soon as possible. AFCSC will publish the VR with a clear explanation as to the interim nature and limitations of those countermeasures. If AFCSC doesn't have the expertise in-house to develop a particular countermeasure, they will at least oversee and coordinate the development and deployment of countermeasures as needed.

4.6 Research and Development

If a vulnerability has been identified and validated by AFCSC and a validated threat does exist, then countermeasures must be developed. If those countermeasures require R&D, then AFCSC will sponsor special R&D efforts to develop the required countermeasure. Depending on the nature of the R&D and the potential applications, the validation of R&D requirements may involve other organizations.

The Consolidated Communications-Computer Systems Security R&D Program is directed by AFCSC under projects LEADING EDGE and FIRESTARTER. If a VR is valid and no countermeasures exist, AFCSC will submit or coordinate the submission of an R&D requirement. Other requirements for the R&D program may come as the result of SPAs that identify special countermeasures that have wide applications throughout the Air Force.

5.0 CONCLUSION

As demonstrated by examples earlier in this document, the need clearly exists for a consolidated effort tying together the disciplines of COMPUSEC, COMSEC, and TEMPEST. With the proliferation of communications-computer systems in the Air Force and DOD, the CVRP is needed to organize the use of our limited communications-computer system security resources.

REFERENCES:

AFR 205-16, **Computer Security Policy**, 28 Apr 89

DODD 5200.28, **Security Requirements for Automated Information Systems**, 21 Mar 88

DODI 5215.2, **Computer Security Technical Vulnerability Reporting Program**, 2 Sep 86

NTISSP 200, **National Policy on Controlled Access Protection**, 15 Jul 87

C. Stoll, "Stalking the Wily Hacker", **COMMUNICATIONS of the ACM**, May 1988

C. Stoll, "What do you Feed a Trojan Horse?", Presented to the 10th National Security Conference, Baltimore, MD, Sep 87

ACKNOWLEDGMENTS:

We would like to thank all of our colleagues who commented on the Concept of Operations for the CVRP, the basis for this paper. We especially thank Lt Glyn M. Runnels for his many hours of editing and layout work for both this paper and the Concept of Operations for the CVRP.

TRACK D

UNETHICAL "COMPUTER" BEHAVIOR: WHO IS RESPONSIBLE?

by LARRY MARTIN

Executive Secretary
Subcommittee on Automated Information Systems Security (SAISS)

Background

We live in a society that clearly believes that maturity and responsibility come with age. This observation is supported by many of our statc and federal laws which establish legal age for certain rights and privileges. Some examples are:

1. Drive
2. Vote
3. Drink Alcohol
4. Enlist in Military
5. Marry
6. Serve as President of the United States

Parents also establish age limit restrictions on their children. They may require them to be at least 16 before they can go out on a date in a car. They may specify in their Last Will and Testament that the children must reach age 25 before they receive any money held in trust for them. The bottom line is that parents and society feel a high degree of confidence that the individuals we've entrusted with certain privileges or resources will behave in a mature and responsible manner when they understand and appreciate the value of a resource or the possible ramifications if a privilege is abused or misused.

The advent of computers has created a paradox for our society. We view computers as an incredible learning tool and something our children must master in order to be successful in the future. In order to give our children a head start, we are introducing them to computers as early as possible. We give them access and the knowledge to operate a very powerful tool that can be potentially damaging if used irresponsibly or in an unacceptable manner. The paradox for our society is that we have not ed an age restriction to the operation of computers. While we would consider it unconscionable to sell a handgun to a 12 year old, or put a second grader behind the wheel of a motor vehicle, we do, in fact, sell computers and modems to 12 year olds and put second graders behind the keyboards of computers.

Thus, we have a dilemma. Do we give our youth a head start toward the future or do we establish an age limit for the purchase and use of computers to promote responsible and ethical use? What a choice - either to enhance the learning abilities of our young or to create a nightmare for law enforcement personnel?

The Educational System

As early as possible, we teach children the basic rules that define acceptable behavior. These standards of conduct include such things as respecting others, keeping our hands to ourselves, not taking things which do not belong to us, and cleaning up after ourselves. They become the simple foundation upon which the rules of acceptable or ethical behavior for mature adults are built. This is the premise of a recent best-seller All I Really Need to Know I Learned in Kindergarten by Robert Fulghum. We start out with simple and basic rules, then build as the mind becomes more capable of comprehension.

As we have brought computers into the elementary school classrooms and begun teaching these young impressionable minds about them, we have been negligent. We have ignored the teaching, as early as possible, of those simple and basic rules that define acceptable or ethical "computer" behavior. These standards of conduct include such things as respecting other computers, keeping our computer's "hands" to itself, and not taking data which do not belong to us. They become the simple foundation upon which the rules of acceptable or ethical "computer" behavior for mature users are built.

We must develop audio-visual tools and techniques that clearly demonstrate at an elementary level the harmful effects of unacceptable "computer" behavior on others and possibly on ourselves. I don't claim to know what these techniques should look like, but I encourage the educators to allocate some research and development funding to start finding the answer. As we endeavor to teach our children the many positive and useful applications of computer technology, we must delicately demonstrate cause and effect, and instill in these young minds that computer users are both responsible and accountable for their "computer" behavior and the effects that they cause, the same as they are responsible and accountable for their everyday social behavior.

Just as we have begun a grass roots effort in the elementary schools to change other undesirable aspects of our society's behavior with "Say No To Drugs" and "No Smoking" campaigns with the hopes of a future drug-free and smokeless society, we must plant the seeds of acceptable "computer behavior" into the elementary school curriculum to begin building the foundation for professional ethics. We must show that the consequences to the individual using a computer unethically are NOT more favorable than the consequences of not using a computer unethically.

Video Game Vendors

Since computers have not been in schools for very long, I speculate that most of us in today's workforce and most of the students in our nation's colleges had our first experiences with hi-tech electronics with video games.

Think about the first time you played Pac Man or Space Invaders. There were no rules posted showing how to play the game. You simply dropped your quarter in the slot and the game started. You learned the rules of the game by trial and error, pushing levers and pressing buttons. In fact, a recent stroll through the video arcade at the local shopping mall confirmed that the same thing is still true. The games are much more sophisticated with more spectacular graphics, but very few of the games have posted rules or instructions.

The vendors of the pay-as-you-play games have a strong motivation for this strategy. The longer it takes a player to learn the rules of the game, the more quarters the player will deposit attempting to conquer the machine and meet its ultimate challenge. Challenge is a very strong motivator and it becomes addictive. It's a scary thought, but the profit strategy of some of these video game vendors is not too unlike the profit strategy of the drug dealer. Once you get the person addicted, the profits come rolling in!

A segment on the ABC News Magazine show "20/20" featured one of the most recent video game crazes, Super Mario Brothers and Super Mario Brothers II by Nintendo. More classic examples of trial and error would be hard to find. The game is full of surprises and most are discovered by accident. The challenge is so great that often the lessons learned by a player the previous day dominate conversation on the school bus or at the lunch table in the school the next day. There is even a newsletter that describes the discoveries of others.

For Christmas 1986, a game was marketed called "Hacker." There were no rules. You simply inserted the disk, got a blank screen, and proceeded by trial and error to break into the "system." Was it a game or a tutorial?

While their motive may be profit, I believe that the vendors of these games have a responsibility to look at their role in influencing the behavioral development of their users to determine if any modifications to their product strategies are appropriate.

The Computer User

The ultimate responsibility to behave in an acceptable manner belongs to the user. For those never taught computer ethics or for those who choose to ignore them, I contend that this trial and error process, which the video game players have become accustomed to, evolves into the mindset that whatever the games allow you to do is within the rules and is, therefore, acceptable or "ethical." This mindset can and does carry over to real computer systems.

As an example, I cite the case of Neal Patrick, the teenage member of the 414 hacker club in Milwaukee that was named after the local area code. Club members penetrated numerous computer systems including the Los Alamos National Laboratory and the Sloan-Kettering Cancer Institute. The club members were all taking computer courses in the high school and had the system manuals for their school's computer.

Upon connecting to another computer via dial-up, the club members would identify the computer as the same as the one in their school from its welcome banner and log-on prompt. Using the system manuals, they proceeded to enter the preset system passwords which are in the system for installation by the Customer Engineers. The vendor system manuals and most installation policies recommend and encourage the changing of these preset passwords immediately upon acceptance of the installation. However, for the systems that the 414's penetrated, these passwords were never changed so the system not only allowed the hackers access but gave them special Customer Engineer privileges.

When asked by a Congressman, under oath, at what point he realized that what he was doing was wrong, Mr. Patrick responded, "when the FBI was knocking at my front door."

Systems Managers, Developers & Security Officers

If my conclusions and assumptions are correct, then much of the responsibility for prevention and detection falls solidly on the shoulders of the system manager, developer and/or security officer. These officials must discourage and prevent any user, authorized or not, from abusing or misusing the system. When, however, prevention is impossible, then detection is a must. It is a given that if these management officials are negligent or overlook something as simple as changing the preset passwords, then the hacker or the authorized but unethical user is likely to exploit it.

I personally observed this phenomenon when I was an employee of another Federal Government Agency over ten years ago. An employee in a local office found that the system allowed her to make corrections for claimants who were erroneously indicated as deceased. When a claimant would come into the office to find out why they had not received their last benefit check, she would access their record. On occasion, she would discover that due to a data entry error, the system had terminated their benefits because the claimant was deceased. She would enter a special "resurrection transaction." The transaction would issue a one-time retroactive check back to their "date of death". After executing a number of authorized "resurrection" transactions, the employee devised a very clever scheme. She went to cemeteries and looked for people who had been dead for at least five years. She took her list of names back to the office and proceeded to execute simultaneous resurrection and change of address transactions. The one)time retroactive checks were sent to her Post Office box. Each day she would query the system, checking for the code that indicated the check had been cut and was in the mail. Once she saw that code in the record, she'd "kill" the person off again and change the address back to what it had been. If the person had been collecting $300 a month and had been dead for five years, these checks would be for substantial amounts of money. She was doing what the system allowed her to do about two or three times per month.

Why did the system allow her to do this? Unfortunately, the system developers had overlooked something simple. There were no thresholds established in the system that required supervisory intervention when a specified period of time or dollar amount had been exceeded. What is the likelihood that someone is going to come in and complain that they haven't received their last 60 benefit checks? The system developers were obviously very sharp to build into the system a way to resurrect and retro-actively pay those who were erroneously deceased. However, they did not conceive of the particular scenario, and therefore did not build in the necessary controls that would prevent the transaction from executing when the time period or dollar amount was unreasonable. The employee simply did what the system allowed her to do.

Another example of a user doing what the system allows him or her to do involved an employee of another local office who had the authority to waive overpayments resulting from a claimant's annual redetermination. This employee would, after hearing a claimant's justification for a waiver, deny the request and set up a cash payment plan for the claimant to repay the overpaid amount. The employee would accept an initial cash payment and issue a phoney receipt to the claimant. After the claimant left the office, the employee would enter a waiver transaction into the system clearing the overpayment amount off of the books. As a steady stream of dutiful claimants came to this employee's desk with their payments, the employee simply pocketed the cash. This is another example of insufficient controls and a user doing what the system permitted. The situation was corrected with a system generated notice to the claimant that could not be suppressed whenever an overpayment was waived. The notice would inform the claimant that they did not have to pay the money back.

System managers, system developers and security officers are human and, like everyone else, make technical and judgmental errors. We are all familiar with Murphy's Law. If it's possible for an operational system to have a flaw in it, then it will. They must recognize that they may, in many cases, be dealing with users, both authorized and unauthorized, who are of the mindset based on a false assumption that what the system allows them to do is acceptable.

One answer for dealing with the user's false assumption is to eliminate it. The posting of rules or electronic "No Trespassing" signs might serve to caution all users at the beginning of their session. "No Trespassing" signs, as we know them, protect physical property and generally cite a law and issue a strong warning as to possible penalties and/or impending danger, such as the "use of deadly force." While there is no pending physical danger with trespassing into computers, if the user does find a flaw in the system's security, he or she would have already been put on notice that certain behavior is unacceptable and perhaps unlawful. If such notices also cited appropriate sections and paragraphs of applicable Federal and State laws, they would also be aware that they might be prosecuted and could be subject to fines and/or imprisonment. There could be no false assumption that whatever the system permitted was fair game and, if ever prosecuted, intent could be more easily established. There are steps that the system managers, developers and security officers can and should take. They must share in the overall responsibility.

Assuming that the system managers, developers and security officers take the necessary steps, we are eliminating as much abuse and misuse as possible and hopefully detecting the rest. However, detection means that these people will get caught. Should our society make these people instant celebrities and glorify what they did or should we punish them and let that serve as an example to others? What responsibility does our society and the news media have in the deterrence or proliferation of unethical "computer" behavior?

I previously referred to the member of 414, Neal Patrick. In addition to testifying before Congress, something very few of us get to do in our lifetime, Mr. Patrick also got his picture on the cover of Newsweek Magazine, something even fewer of us achieve in our lifetime. And as the story goes, at that time the movie "War Games" was popular and, as a publicity stunt, a local theater manager where "War Games" was showing, paid Mr. Patrick to sign autographs for movie-goers in front of the theater. All in all, quite an accomplishment for a 17-year old. As a model for other teens, this incident was more of an incentive than a deterrent.

Society

Again, we as a society have a paradox. While it is traditional of our society to bestow punishment befitting the crime, the criminals in most of these cases are young intelligent students who represent some of the most brilliant minds in our country. It is these brilliant minds that we rely upon to carry this country into the 21st Century.

Do we lock up these young intelligent students in prison with murderers and bank robbers or do we find a way to channel their endeavors into more positive and productive activity? Do we deny these people a college degree when there are condemned murderers on death row earning college degrees?

We must, as a society, recognize that we have a new criminal element that has some of the familiar characteristics, but is in a class by itself. In the same way that the use of computers has required change to the way we as a society function, so too, has the misuse of computers required change to the way we as a society deal with those who operate outside the established rules. Are we making these changes timely and proactively or waiting for the crimes to occur and then reacting?

News Media

I believe the news media shares in the responsibility in helping to deter computer crime, misuse and abuse. I believe it would have been more of a deterrent if the cover of Newsweek would have shown Mr. Patrick being placed in the backseat of a police car with his hands cuffed behind his back while other officers carried his computer equipment out of his house instead of depicting him with a smug look as if to say "I showed you!"

Last November, when the Arpanet Worm brought the network down, every television network and the front page of every major newspaper and magazine carried it as their feature story for days. One would have to have been locked away in a monastery to have not heard about it. Yet on February 15, 1989, the day after Herbert Zinn, Jr., the 18-year old hacker who penetrated AT&T and NATO systems, was convicted and sentenced to a year in prison and a $10,000 fine, a one paragraph article appeared on the front page of the Business Section of the local newspaper. Mr. Zinn was the first conviction under the new Federal Law, known as the Computer Fraud and Abuse Act of 1986. A true landmark in the computer security chronology and it went practically unnoticed. Highlighting prison sentences, fines and other negative results experienced by the perpetrators as well as the damage, pain and suffering of the victims of computer crimes should at least make a potential hacker or computer criminal think twice and examine the risks before acting.

Victims

The victims of computer crimes often do not prosecute for fear of the adverse affects of the negative publicity. This failure to prosecute contributes to the temptation of would-be computer criminal because it lessens the fear of reprisal. For the individual who may be contemplating a computer crime, it may make the difference between right and wrong. Therefore, I add victims to the list of those responsible for deterring the computer criminal.

Parents

However, the list is not complete without including parents. Earlier, I mentioned a game marketed during Christmas 1986 called "Hacker." Some parents, no doubt, gave this "computer game" to their children for Christmas. Those who got the game for Christmas had the next six months to become proficient utilizing this hacker "self-tutorial" in order to be ready for their three-month summer vacation from school. Instead of hanging out at the mall or on the street corner, they could simply spend their days and nights in the comfort of their own bedrooms.

With the many everyday pressures from job and family, many parents might relax and find comfort in knowing that their child is nice and safe in their bedroom quietly "fooling around" with their computer, instead of getting in trouble by hanging out with the wrong crowd. What they may not realize is that their child may be attempting to penetrate any computer their modem might bring them in contact with. Certainly Mr. Patrick's parents and Mr. Zinn's parents were unaware.

If a child had a gun or other deadly weapon in their possession, then their parents would surely want to know about it. Why should it be any different with a modem? Many teenagers have their own money and have the ability to make their own purchases. So it is not unreasonable that a child could have a modem and the parents not know. There may be many parents who are not computer literate and even if they knew their child had a modem would not know what a modem is and what it does. Should we require permits for modems? Should we restrict the sale to those 21 & over unless signed for by a responsible adult after that adult has read the risks associated with having a modem in the home? The bottom line is that parents have a responsibility to know what their children are doing and to help the children understand what is acceptable behavior and what is unacceptable.

We've all heard the cliche "Do as I say and not as I do." We teach our children by example. They tend to emulate our behavior. Our hi-tech world makes it very easy for us to serve as bad examples for our children without us even thinking about it. How many of us have copied a video tape on our VCR's even though it has an FBI warning right in the beginning? How many of us have made bootleg copies of copyrighted computer software? Why did we do it? Simple, because we didn't want to pay for it. If we walk into a store and take something and walk out without paying for it, we are stealing. But if we copy a video tape or computer software, we do not think of it as stealing. Perhaps it is not perceived to be as blatant, but someone ultimately suffers from the loss of the proper purchase of whatever we copied. But surely the loss of one measly sale will not bankrupt a large corporation. Are our ethics now contingent upon the net worth of the potential victim? Have we then become a modern day electronic Robin Hood? We take from the perceived "rich" (owner of the copyright) and give to the perceived "poor" (ourselves or our children). Although we act with the best of intentions as loving parents who want our children to have something that they or we may not otherwise be able to afford, the intangible repercussions of our actions may be ultimately hurting them. When our children observe this behavior, at the very least they may be confused by what appears to be a double standard. It's very similar to the confusion of children of parents who smoke. The schools are teaching that smoking is unhealthy for the smoker and those around him or her. When children see a parent smoking, they must either think that the parent is unaware of the risks or must think that the parent doesn't care about his or her own health and the health of those around him or her. If they try to tell the parent of the risks, it is doubtful such a reprimand from child to parent would be well received. They might just think well Mom or Dad isn't worried about it, so why should I? How can that p ›rent convince the child not to smoke after teaching by example?

Who can say how far they will take the examples we set for them today, as technology improves? Things we cannot even dream of today will be achievable 20 years from now when our teenagers are in the workforce. We must be conscious of our own actions and realize that if we exhibit unacceptable behavior, our children are likely to do the same. We are setting a precedent for society. So yet another piece of the ethics puzzle is the parents, who must recognize their role in the ethical development of their children. I believe that we as a society need an awareness raising as to the threats to our ethics by technology. We need to examine these subtle capabilities made possible by technology that allow us to deviate unconsciously from our normal ethical behavior. We may also need to raise the public awareness of computer cause and effect much like the one described earlier for the elementary schools, but of course, scaled accordingly to the appropriate level of comprehension. Raising public awareness is another possible role for the news media and the educational system in the overall scheme.

Conclusion

While the schools, the video game vendors, the systems managers, developers and security officers, the news media, the victims, the parents and society as a whole share in the responsibility for the computer user's behavior, the ultimate responsibility to behave in an acceptable .manner belongs to the user.

There are many parallels between acceptable social behavior and acceptable "computer" behavior as illustrated in the following table:

UNACCEPTABLE SOCIAL BEHAVIOR	UNACCEPTABLE "COMPUTER" BEHAVIOR
1. To knowingly infect another person with a communicable disease	To knowingly infect another person's computer with a virus or worm
2. To enter another person's home or drive another person's car without their permission	To enter another person's system without permission
3. To rummage through another person's belongings	To rummage through another person's database(s)
4. To shoplift or steal something that belongs to another	To make copies of copyrighted software
5. To keep the extra money if a store clerk gives us back too much change or to pay a lower price because merchandise is priced wrong	To access another's system or data because the system allows us to
6. To lock another person out of his or her own household or car	To deny someone the use of his or her computer

It takes a long time to change society's behavior. Drunk driving is a perfect example. With all of the attention it has received in recent years, the increased penalties, and the increased police activity such as sobriety check points, there were still over 34,000 drunk driving arrests in Maryland in 1988. That's nearly 100 per day or one every 15 minutes. While the number of drunk driving arrests is on the decline, one every 15 minutes is a very real indicator that the problem has not gone away. And keep in mind that the 34,000 were only the ones that got caught.

I believe that drunk driving and computer misuse are the crimes that will reflect the 80's. It was this decade that raised the public awareness to the negative repercussions of both and the parallels are quite interesting. They both represent behavior which was tolerated until the potential dangers were realized. As awareness grows, tolerance lessens. Behavior that was once considered prankishness or mischievous is starting to be considered malicious and a criminal offense. How to deal with these new type criminals who are not typical of the stereotype criminal element has become an issue.

In Maryland, there was a recent proposal to build a separate correctional facility just to house convicted drunk drivers who are given prison sentences. Our prisons are already overcrowded and adding 34,000 drunk drivers would only add to the problem. Such a facility would not only prevent the strain on the prison system but would segregate the drunk driver from hardened criminal.

While many consider the computer security problem and solution to be technical, I believe that the computer security problem is a people problem with both a technical and a people solution. As we all know, it is not a perfect world and the teaching of "computer ethics" will not eliminate unethical computer use or the user who puts himself or herself above the law. There will always be some who, although they know the rules, will disobey them. Our prisons are filled with former students who were taught the rules of ethical social behavior and somewhere along the way, have chosen not to obey them. As more and more laws are enacted that make specific acts of unethical computer use unlawful, it will become more and more difficult to distinquish between computer ethics and computer crime.

If the computer user behaves in a unethical manner, then the technical solutions must be there. It is only when we can't trust the behavior of the computer user, that we must have mechanisms and assurances that allow us to trust the system instead. As I have tried to describe, all of us share in the responsibility for and the consequences of unethical computer use.

MALICIOUS CODE: AN ETHICAL DILEMMA

Maj. (Select) Glenn D. Watt, Jr., C321
National Computer Security Center
Fort Meade, MD 20755

Introduction:

In the early 1980s, the city of Cambridge, Massachusetts, voted to petition Harvard University to temporarily halt the construction of a very expensive laboratory for specialized genetics research. This action, initiated and supported by distinguished members of the faculty, recognized the potentially dangerous situation at hand. This example is typical of what professionals usually do when they encounter an immature technology. The information about the atomic bomb and other such devices also was tightly controlled by military professionals with an ethical standard that demanded control to assure the protection of the larger community. A technology equally dangerous to the national computer security community is malicious code. It is a problem that has crossed international borders, and threatens the integrity of every type of system from personal computers to super computers. In 1985 J.M. Carroll and H. Juergensen performed mathematical proofs showing that any current state-of-the-art time sharing, multiprogramming environment could not simultaneously support security and integrity without compromising protection, efficiency or both [1]. The National Computer Security Center's (NCSC) Trusted Computer System Evaluation Criteria (TCSEC) and Trusted Network Interpretation (TNI) guidelines do not specifically address viruses. In fact, the Internet Virus of 1988 might have propagated on a B2 system and perhaps even on an A1. Will technology alone solve the problem of malicious code? If not, how should we then compute?

The Problem:

Malicious code can take the form of a virus, worm, Trojan horse, logic bomb or time bomb. No matter what the form, each piece of code needs a transport mechanism to move from one system to the next. In the past, most malicious code resided on bulletin board systems (BBS) or portable magnetic media. This would require an explicit download from a BBS or insertion of a previously contaminated floppy disk. Computer networks, however, have made this form of transportation obsolete. Malicious code now can be written and injected into the mainstream of computing without any human action required. The perception of the threat from malicious code is somewhat analogous to the past development history of the atomic bomb. Originally the atomic bomb needed a rather large bomber and bombers could be shot down before they reached their targets. In 1957 the Soviets proposed the idea that guided missiles could be used instead of bombers. Then they launched Sputnik and demonstrated the potential capability. Now the world had a much more difficult situation because delivering the bomb had suddenly become easier and faster. Similarly, there was a period of time when computer professionals did not consider malicious code as serious a threat because the available transport mechanisms limited the speed and to some extent the amount of damage. In the computer security world now, however, malicious code is our atomic bomb and networks our guided missiles.

Should we freely and openly research this area in order to solve the problems of today, or limit open research until we better understand the situation and produce effective countermeasures. Perhaps a part of the answer lies outside of a technological framework and in an ethical one. Let's examine the issue of malicious code and what can be done to solve the problem.

The Effects

Three fundamental effects - economic, technological and psychological - form the foundation for a discussion on the results of executing malicious code.

Economic Effect: The economic influence of malicious code can be broken down into three basic components: checking for damage, analyzing the malicious code, and developing or installing fixes. Checking for damage can be no small chore. All systems software and associated data must be checked immediately. On a typical system that could take anywhere from hours to days, with eight hours being a good average. After sanitizing the system software, verification of user programs begins. Although usually accomplished by the end user, the effect is still the same - wasted time. After damage assessment, analysis of the code and installation of new safeguards begins. During a post mortem meeting on the effects of the Internet Virus of 1988, attended by government and academia, the cost of that malicious code incident alone was estimated at $2,000,000. Congressman Wally Herger, coauthor of H.R. 5061 "Computer Virus Eradication Act of 1988", sent a letter to the Information System Security Association stating there may have been 2500 malicious code attacks to date at a cost of $20 million.[2] The economic effect is real and is not cheap!

Technological Effect: Malicious code also affects technology in terms of software. Software such as the UNIX operating system grew to its current state largely because of the availability of source code in the early days. Source code was readily available throughout the V6, V7 and PWB UNIX releases. Many universities and research organizations modified it, shared it, and in a sense became responsible for its debugging and maturation. Along with the benefits of having open software came the requirement to use the information responsibly. Unfortunately ever since it's inception people have been trying to break UNIX systems. The reason is largely due to the fact that source code to the operating system was readily available and used in numerous universities to teach systems fundamentals. With an intimate knowledge of the operating system security attacks are greatly simplified. In response the UNIX community made the operating system more secure by controlling the distribution of source code, and by implementing security features such as limited "root" login access. To some degree, control is due to the vendors' desire to standardize; however, security is playing a major role. However, it is distressing that future computer scientists may not have the learning experience of pouring over the source code to a functicning operating system while in school. Perhaps the cost of training our next generation of systems software gurus will have to be born by industry and government after hiring.

On the positive side, the technology of state-machine models is actually improving because of the threat of malicious code. The partial ordering known as simple security and the confinement property, set forth by Bell & Lapadula, established a provably secure mathematical model in 1973. Although still used as one model from which to develop secure systems, malicious virus code is promoting fresh looks at the model. The question must be asked if A≥B "A dominates B" doesn't actually promote the spread of viruses from lower security levels to higher ones. Malicious code is not only forcing this question to be asked, but also the correction of the security model if it is found to be flawed.

Psychological Effect: The psychological effect is perhaps the most profound and yet the least addressed. According university reports and this authors own experience, systems administrators, computer center directors and end users all become paranoid after a malicious code attack. Systems that may have gone for months without a backup suddenly receive undivided attention. Administrators who reveled in living dangerously become the vanguards of security. It has been said, "In order to assure a person gets the message, advertising has to be memorable."[3] Is there any more memorable way to get the security message across than to be the victim of a malicious code attack.

The use of an attack can best be described by the analogy of having a home broken into and robbed. Before the event, the residents feel safe and secure behind the locked doors and windows. After the event, shock sets in. The security factor vanishes in the stark reality that locks can be broken and windows opened. Most will install better security devices and fix the holes that are now apparent, but some, albeit a small percentage, will leave the area never feeling safe there again. In the computer security arena the same attitudes surface.

Most computer sites will recover from a malicious code attack, implement tighter security features and press on. Some, however, will not recover. They will restore their systems and decide that it just isn't worth being on a network, or using software of unverifiable origin. Although the cost, intellectually and financially, may be great they will not risk another attack. In this case the perpetrator has inflicted psychological damage.

The Internet VIRUS of 1988 provided an excellent example of the psychological effect. Soon after the detection of the virus major sites throughout the net dropped off. Some managers went so far as to shut down the servers and actually pull the plugs! The result was devastating, but continued for an extended period of time. Even after many of the sites did return, many gateways were off. Now, managers should not be saying "damn the torpedoes (or viruses) full steam ahead." Quite the contrary, quarantine is a good approach to stop a virus. The psychological problem arises with sites that choose never to return, cancel plans to connect, or severely modify their functionality within the system. For example, one site has stopped receiving mail as a security precaution. In another situation, system managers implemented extreme measures to make sure their software was virus free.

These varied and far reaching influences are also steering the computer community toward a more permanent solution. That solution will involve both technology and ethics.

The Solution :

-- Secure Computers and Computer Networks:

A former Director of the National Security Agency, Lieutenant General USAF (Ret) Lincoln Faurer recently stated that "Only recently, with the advent of media reports about computer viruses and program tapeworms, have computer security issues taken on a higher and more appropriate visibility."[4] The Computer Security Act, signed by President Reagan early in 1988 provides another example of our society's growing demand for professional protection. The millions of computer users, growing at a rate of roughly 70% annually, are rapidly demanding protection. Admittedly, legislation, when used in conjunction with ethical leadership, supports an effective part of the answer, but not the entire answer. Secure computers and computer networks will play an important role in solving the malicious code security problem. Government and private industry are looking into secure network components for both local and long-haul networks. Research and development in this area must not only continue, but increase. New technologies that are developed and manufactured as a direct result of research, alongside well established data encryption, will provide a broad base of protection. The problem of the next decade, systems integrity and denial of service, will require systems that are secure both in data confidentiality and operation. Applying systems integrity and denial of service to computer networks turns a two dimensional problem into a three dimensional one. The problem has been portrayed as a bucket brigade trying to put out fires in several modern high-rise buildings. Fortunately, a great deal of work is currently being done in this area. A quick review of the proceedings of any security conference will verify just how much is being done in the technological part of the solution.

-- Ethical Leadership:

According to a 1977 issue of the *Harvard Business Review*, legislation is an important part of influencing business practices, but ethical codes would have a greater impact on executives and corporations.[5] This is the other side of the issue. As professionals we must take an ethical stand and set an example for others to follow. Since the world is becoming increasingly dependent on computers and computer networks, we need to help in the establishment of a workable standard of ethics. Mr. Harry B. DeMaio, Information Security Products Manager for Deloitte Haskins & Sells, recently said, "The organizations to which we normally look for ethical leadership - church, school, government, home, the media - lack the technical knowledge, the budget, and even the awareness to deal with this subject in the electronic world of today and tomorrow." [6] Perhaps, because so few professionals have tried to combine both computer science and philosophical ethics, so little work has been done in this area. Nevertheless, it is imperative to develop a workable, consistent standard from which to operate. There are several steps that should be taken. First, if we are serious about the need for computer security, educating young engineers and scientists about the unacceptable ethics of exploiting weaknesses in computer systems or networks for financial gain or personal satisfaction must be a priority. College, and perhaps even high school, is an appropriate place to start educating our future engineers. Harvard Business School already has adopted this priority by announcing that all MBA students must take a 3 week course in ethics.[7] Most universities are requiring students to take some form of a computer course as a graduation requirement. Computer literacy is the desired goal, with some schools requiring a beginner's knowledge of programming. If a university devoted several classes during the course to computer ethics, perhaps the "wily hackers" of the campus crowd would be reduced. Having students simply study several existing codes, such as the ones included in this paper, would provide a basic framework about the behavior expected of computer professionals. For computer science and engineering majors, most universities encourage the students to experiment and expand their understanding on the hardware and software. There is nothing inherently wrong with this, unless encouraged without an ethical framework by which to judge what is right and what is wrong. Without that framework, the student soon discovers that non destructive malicious code can be a vehicle to personal recognition. The perpetrator, neither intending to nor actually destroying data, assumes no harm is done; however, because of the previously mentioned effects, that is simply not true. The cost is non zero and is indeed higher than most people, and some professionals would expect. A portion of these costs are a direct result of inadequate standards of conduct.

Second, an ethical standard of conduct must start with ethical leadership. It begins with management and works its way down to the grass root engineer by enforcing what we already know to be proper. For example, how many sites do you know of that have illegal copies of software. If we can't even keep our own shops honest, why should we expect that of anyone else? In this example, a law governs what is right and what is wrong. Laws are a good place to start, but they only provide a minimum standard that must be adhered to. An ethical standard of conduct must go beyond the law. For example, considering it unethical, a surgeon will usually not operate on a family member. Under the law, both relative and non relative are equal, but the ethical standards by which the surgeon operates requires the physician to restrict practice when it comes to family members. As a medical student, the future physician attends classes on medical ethics. As an intern, he gets on-the-job reinforcement of those ethics from older doctors. At some point the physician will, in turn, influence younger interns to adopt the medical ethics also. As computer scientists, we seem to avoid such non scientific issues. The computer science community has taken the time to write down codes of ethics. Now it is time to emphasize these codes in the workplace. Since disobeying an ethical code is not important until people accept that code as a standard by which to live, we need leaders who will teach and reinforce standards of conduct for computer professionals.

Third, professional societies, universities, government, industry, and religious institutions need to help in reviewing, and upgrading existing codes to make them applicable and workable today. Over the years several good codes have been established[8], however, when they were drawn up, malicious code was for the most part non existent. The Data Processing Management Association Code of ethics (Appendix A), provides some of the strongest standards anywhere. Its members are encouraged by their obligation to society to protect the privacy and confidentiality of all information, insure that products are used in a socially responsible way, support, respect and abide by the appropriate local, state, provincial and federal laws, not use knowledge of a confidential or personal nature in any unauthorized manner or to achieve personal gain. As an obligation to the employer, the member should not exploit the weakness of a computer system for personal gain or personal satisfaction. This code was endorsed in January of 1983. Some older codes of ethics, like the ACM and the IEEE standards aren't as strong in the area of malicious code. This is not to say that their codes don't promote ethical computing. Both the ACM and the IEEE codes of ethics encourage their members to practice computer science and engineering in a dignified, professional manner. A review of these codes will show that the primary concern of each code of ethics was misrepresentation by its members to their employers and clients. Some preventative maintenance on these codes of ethics could bolster a professional attitude towards malicious code in a world that now encompasses personal computers, supercomputers and networks of computers.

Professional societies can develop stronger standards to encourage the regulation of a computer's use. They need to emphasize that research and experimentation is good, but doing it for the purpose of breaking security codes, denying service to other users, or somehow compromising system integrity should be strongly discouraged. Establishing a code will not assure compliance nor acceptance by every member but the society in general will need to accept and promote the code before peer pressure will make it effective. "An ethic is esoteric until it is put into practice."[9] The Data Processing Management Association, ACM, and IEEE all have a good base from which to work, but developing an ethic is not the sole task of any one professional society. Ideas, suggestions and guidance must also come from universities, government, industry and religious institutions.

Government and industry can begin to promote the development of specific ethical standards for their computing employees. These ethical standards could be periodically emphasized in much the same way as EEO and sexual discrimination ethics are today. Government and industry also might follow the lead of Arthur Anderson & Co.[10] who is funding a five year $5 million effort to promote and assist in getting ethics courses into graduate and undergraduate business schools. If government and industry could promote similar programs for science and engineering students schools would more amenable to offering computer ethics as part of a curriculum.

Churches can provide a source of direction not usually considered. Throughout history religious institutions have dealt with ethics and society. A study of history will show that religious leaders had answers to societal problems derived from a totally different source. Often they had the answers to injustices when no one else did. Unfortunately society had and has a tendency not to listen to them, because social problems aren't religious in nature. In retrospect, today we see that they really did understand the implications of a society's code of ethics. Church leaders have dealt with numerous ethical issues and should be consulted to examine the issues and provide input for computer ethics. An understanding of how malicious code affects the psychological aspects of another human being would be a good start for this institution. From an understanding of the effects, ethical codes could be written to deal with the cause. There can be no doubt that computer-based information is the new raw material of our present and future society. We must involve all elements of society in its safeguarding.

Conclusion:

In the final analysis computer professionals should recognize that ethical standards are equally important as technology when it comes to computer security and malicious code. An attack must be waged on two fronts. An interdictive ethical attack needs to mounted as soon as possible to change attitudes. A change in computing ethics would weaken the supply line of new malicious code writers. In parallel the technological efforts, which have been ongoing for some time now, must be fortified. A Pentagon commission report stated that research in the area of security was in a deplorable state, while at the same time others like Dr. Cliff Stoll emphasis that effective security must rest on a foundation of research.[11] In a broader sense if research is the foundation of security, than ethical computing is the mortar that holds it all together.

References

[1] J.M. Carrol, and H. Juergensen, Design of a Secure Relational Database, *Proc IFIP/SEC 1985*, pp. 1-15

[2] Wally Herger, Member of Congress, *ISAA Access*, Vol 2 Issue 1 p13.

[3] Dennis Poindexter, Security Awareness: Making It Happen, *Proc. 11th National Security Conference*, October 1988

[4] Lincoln Faurer, Building Secure Worldwide Communications Networks, Datamation Special Edition on Computer Security Issues & Trends

[5] Harvard Business Review, January-February 1977

[6] Harry B. DeMaio, The Information Ethics Issue: It's Time for Management Action, *Datamation Special Edition on Computer Security Issues and Trends*

[7] Edwin B. Heinlein, Corresponding Committee on Law and Ethics, ISAA Access, Vol 2 Issue 1 p15.

[8] DPMA Code of Ethics, (Appendix A), ACM Code of Professional Conduct (Appendix B), IEEE Code of Ethics (Appendix C)

[9] Douglas W. Johnson, Computer Ethics - A Guide for the New Age, p 115.

[10] Edwin B. Heinlein, Corresponding Committee on Law and Ethics, *ISAA Access*, Vol 2 Issue 1 p27

[11] Dr. Cliff Stoll, How Secure are Computers in the U.S.A. - An Analysis of a Series of Attacks on Milnet Computers, *Computers & Security* December 1988

Attachment A

DPMA Code of Ethics, Standards of Conduct and Enforcement Procedures
Data Processing Management Association

Code of Ethics

I ACKNOWLEDGE:

That I have a obligation to management, therefore, I shall promote the understanding of information processing methods and procedures to management using every resource at my command.

That I have an obligation to my fellow members, therefore, I shall uphold the high ideals of DPMA as outlined in its international bylaws. Further, I shall cooperate with my fellow members and shall treat them with honesty and respect at all times.

That I have an obligation to society and will participate to the best of my ability in the dissemination of knowledge pertaining to the general development and understanding of information processing.
Further, I shall not use knowledge of a confidential nature to further my personal interest, nor shall I violate the privacy and confidentiality of information entrusted to me or to which I may gain access.

That I have an obligation to my employer whose trust I hold, therefore, I shall endeavor to discharge this obligation to the best of my ability, to guard my employer's interests, and to advise him or her wisely and honestly.

That I have an obligation to my country, therefore, in my personal, business and social contacts, I shall uphold my nation and shall honor the chosen way of life of my fellow citizens.

I accept these obligations as a personal responsibility and as a member of this association. I shall actively discharge these obligations and I dedicate myself to that end.

Standards of Conduct

These standards expand on the Code of Ethics by providing specific statements of behavior in support of each element of the Code. They are not objectives to be strived for; they are rules that no true professional will violate. It is first of all expected that information processing professionals will abide by the appropriate laws of their country and community. The following standards address tenets that apply to the profession.

In Recognition of My Obligation to Management I Shall:

Keep my personal knowledge up-to-date and insure that proper expertise is available when needed.

Share my knowledge with others and present factual and objective
information to management to the best of my ability. Accept full responsibility for work that I perform.

Not misuse the authority entrusted to me.

Not misrepresent or withhold information concerning the capabilities of equipment, software or systems.

Not take advantage of the lack of knowledge or inexperience on the part of others.

In Recognition of My Obligation to My Fellow Members and the Profession I Shall:

Be honest in all my professional relationships.

Take appropriate action in regard to any illegal or unethical practices that come to my attention. However, I will bring charges against any person only when I have reasonable basis for believing in the truth of the allegations and without regard to personal interest.

Endeavor to share my special knowledge.

Cooperate with others in achieving understanding and in identifying problems.

Not use or take credit for the work of others without specific acknowledgment and authorization.

Not take advantage of the lack of knowledge or inexperience on the part of others for personal gain.

In Recognition of My Obligation to Society I Shall:

Protect the privacy and confidentiality of all information entrusted to me.

Use my skill and knowledge to inform the public in all areas of my expertise.

To the best of my ability, insure that the products of my work are used in a socially responsible way.

Support, respect and abide by the appropriate local, state, provincial and federal laws.

Never misrepresent or withhold information that is germane to a problem or situation of public concern nor will I allow any such known information to remain unchallenged.

Not use knowledge of a confidential or personal nature in any unauthorized manner or to achieve personal gain.

In Recognition of My Obligation to My Employer I Shall:

Make every effort to ensure that I have the most current knowledge and that the proper expertise is available when needed.

Avoid conflict of interest and insure that my employer is aware of any potential conflicts.

Present a fair, honest and objective viewpoint.

Protect the proper interests of my employer at all times.

Protect the privacy and confidentiality of all information entrusted to me.

Not misrepresent or withhold information that is germane to the situation.

Not attempt to use the resources of my employer for personal gain or for any purpose without proper approval.

Not exploit the weakness of a computer system for personal gain or personal satisfaction

From DPMA Code of Ethics, Standards of Conduct and Enforcement Procedures. This Code includes documents approved at the 1981 and 1982 International Board of Directors meetings and enforcement procedures effective January 1,1983. Reprinted by permission of the Data Processing Management Association.

ATTACHMENT B

ACM Code of Professional Conduct
Procedures for the Enforcement
of the ACM Code of Professional Conduct

Association for Computing Machinery

[*Code*]

Preamble

RECOGNITION OF PROFESSIONAL STATUS by the public depends not only on skill and dedication but also on adherence to a recognized code of Professional Conduct. The following Code sets forth the general principles (Canons), professional ideals (Ethical Considerations), and mandatory rules (Disciplinary Rules) applicable to each ACM Member.

The verbs "shall"(imperative) and "should" (encouragement) are used purposefully in the Code. The Canons and Ethical Considerations are not, however, binding rules. Each Disciplinary Rule is binding on each individual Member of ACM. Failure to observe the Disciplinary Rules subjects the Member to admonition, suspension or expulsion from the Association as provided by the Procedures for the Enforcement of the ACM Code of Professional Conduct, which are specified in the ACM Policy and Procedures Guidelines. The term "member(s)" is used in the Code. The Disciplinary Rules of the Code apply, however, only to the classes of membership specified in Article 3, Section 4, of the Constitution of the ACM

Canon 1

An ACM member shall act at all times with integrity.

Ethical Considerations

EC1.1 An ACM member shall properly qualify himself when expressing an opinion outside his areas of competence. A member is encouraged to express his opinion on subjects within his area of competence.
EC1.2 An ACM member shall preface any partisan statements about information processing by indicating clearly on whose behalf they are made.
EC1.3 An ACM member shall act faithfully on behalf of his employers
or clients.

Disciplinary Rules

DR1.1.1 An ACM member shall not intentionally misrepresent his qualifications or credentials to present or prospective employers of clients.
DR1.1.2 An ACM member shall not make deliberately false or deceptive statements as to the present or expected state of affairs in any aspect of the capability, delivery, or use of information processing systems.
DR1.2.1 An ACM member shall not intentionally conceal or misrepresent on whose behalf any partisan statements are made.
DR1.3.1 An ACM member acting or employed as a consultant shall, prior to accepting information from a perspective client, inform the client of all factors of which the member is aware which may affect the proper performance of the task.
DR1.3.2 An ACM member shall disclose any interest of which he is aware which does or may conflict with his duty to a present or prospective employer or client.
DR1.3.3 An ACM member shall not use any confidential information from any employer or client, past or present, without prior permission.

Canon 2

An ACM member should strive to increase his competence and the competence and prestige of the profession.

Ethical Considerations

EC2.1 An ACM member is encouraged to extend public knowledge, understanding, and appreciation of information processing , and to oppose any false or deceptive statements relating to information processing of which he is aware.

EC2.2 An ACM member shall not use his professional credentials to misrepresent his competence.

EC2.3 An ACM member shall undertake only those professional assignments and commitments for which he is qualified.

EC2.4 An ACM member shall strive to design and develop systems that adequately perform the intended functions and that satisfy the employer's or client's operational needs.

EC2.5 AN ACM member should maintain and increase his competence through a program of continuing education encompassing the techniques, technical standards, and practices in his fields ot professional activity.

EC2.6 An ACM member should provide opportunity and encouragement for professional development and advancement of both professionals and those aspiring to become professionals.

Disciplinary Rules

DR2.2.1 An ACM member shall not use his professional credentials to misrepresent his competence.

DR2.3.1 An ACM member shall not undertake professional assignments without adequate preparation in the circumstances.

DR2.3.2 An ACM member shall not undertake professional assignments for which he knows or should know he is not competent or cannot become adequately competent without acquiring the assistance of a professional who is competent to perform the assignment.

DR2.4.1 An ACM member shall not represent that a product of his work will perform its function adequately and will meet the receiver's operational needs when he knows or should know that the product is deficient.

Canon 3

An ACM member shall accept responsibility for his work.

Ethical Considerations

EC3.1 An ACM member shall accept only those assignments for which there is reasonable expectancy of meeting requirements or specifications, and shall perform his assignments in a professional manner.

Disciplinary Rules

DR3.1.1 An ACM member shall not neglect any professional assignment which has been accepted.

DR3.1.2 An ACM member shall keep his employer or client properly informed on the progress of his assignments.

DR3.1.3 An ACM member shall not attempt to exonerate himself from, or limit his liability to clients for his personal malpractice.

DR3.1.4 An ACM member shall indicate to his employer or client the consequences to be expected if his professional judgement is overruled.

Canon 4

An ACM member shall act with professional responsibility.

Ethical Considerations

EC4.1 An ACM member shall not use his membership in ACM improperly for professional advantage or to misrepresent the authority of his statements.

EC4.2 An ACM member shall conduct professional activities on a high plane.

EC4.3 An ACM member is encouraged to uphold and improve the professional standards of the Association through participation in their formulation, establishment, and enforcement.

Disciplinary Rules

DR4.1.1 An ACM member shall not speak on behalf of the Association or any of its subgroups without proper authority.

DR4.1.2 An ACM member shall not knowingly misrepresent the policies and views of the Association or any of its subgroups.

DR4.1.3 An ACM member shall preface partisan statements about information processing by indicating clearly on whose behalf they are made.

DR4.2.1 An ACM member shall not maliciously injure the professional reputation of any other person.

DR4.2.2 An ACM member shall not use the services of or his membership in the Association to gain unfair advantage.

DR4.2.3 An ACM member shall take care that credit for work is given to whom credit is properly due.

Canon 5

An ACM member should use his special knowledge and skills for the advancement of human welfare.

Ethical Considerations

EC5.1 An ACM member should consider the health, privacy, and general welfare of the public in the performance of his work.

EC5.2 ACM member, whenever dealing with data concerning individuals, shall always consider the principle of the individuals privacy and seek the following:

To minimize the data collected
To limit authorized access to the data
To provide proper security for the data
To determine the required retention period of the data
To ensure proper disposal of the data

Disciplinary Rules

DR5.2.1 An ACM member shall express his professional opinion to his employers of clients regarding any adverse consequences to the public which might result from work proposed to him.

From ACM Code of Professional Conduct and Procedures for the Enforcement of the ACM Code of Professional Conduct, Used by Permission of the ACM.)

APPENDIX C
IEEE Code of Ethics

Preamble

Engineers, scientists and technologists affect the quality of life for all people in our complex technological society. In the pursuit of their profession, therefore, it is vital that IEEE members conduct their work in an ethical manner so that they merit the confidence of colleagues, employers, clients and the public. This IEEE Code of Ethics represents such a standard of professional conduct for IEEE members in the discharge of their responsibilities to employers, to clients, to the community and to their colleagues in this Institute and other professional societies.

Article I

Members shall maintain high standards of diligence, creativity and productivity and shall:

1. Accept responsibility for their actions.
2. Be honest and realistic in stating claims or estimates from available data.
3. Undertake technological tasks and accept responsibility only if qualified by training or experience, or after full disclosure to their employers or clients of pertinent qualifications.
4. Maintain their professional skills at the level of the state of the art, and recognize the importance of current events in their work.
5. Advance the integrity and prestige of the profession by practicing in a dignified manner and for adequate compensation.

Article II

Members shall, in their work:

1. Treat fairly all colleagues and co-workers, regardless of race, religion, sex, age, or national origin.
2. Report, publish and disseminate freely information to others, subject to legal and proprietary restraints.
3. Encourage colleagues and co-workers to act in accord with this Code and support them when they do so.
4. Seek, accept and offer honest criticism of work, and properly credit the contributions of others.
5. Support and participate in the activities of their professional societies.
6. Assist colleagues and co-workers in their professional development.

Article III

Members shall, in their relations with employers and clients:

1. Act as faithful agents or trustees for their employers or clients in professional and business matters, provided such actions conform with other parts of this code.
2. Keep information of the business affairs or technical processes of an employer or client in confidence while employed, and later, until such information is properly released provided such actions conform with other parts of this Code.
3. Inform their employers, clients, professional societies or public agencies or private agencies of which they are members or to which they may make presentations, of any circumstance that could lead to a conflict of interest.
4. Neither give nor accept, directly or indirectly, any gift, payment or service of more than nominal value to or from those having a business relationship with their employers or clients.
5. Assist and advise their employers or clients in anticipating the possible consequences, direct and indirect, immediate or remote, of the projects, work or plans of which they have knowledge.

Article IV

Members shall, in fulfilling their responsibilities to the community:

1. Protect the safety, health and welfare of the public and speak out against abuses in these areas affecting the public interest.
2. Contribute professional advice, as appropriate, to civic, charitable or other nonprofit organizations.
3. Seek to extend public knowledge and appreciation of the profession and its achievements.

INFORMATION SECURITY AS A TOPIC IN UNDERGRADUATE EDUCATION OF COMPUTER SCIENTISTS

John C. Higgins
346 TMCB Brigham Young University
Provo, Ut. 84602

1. Introduction

This presentation will not attempt to argue the relative importance of information security as a formal area of study. It is assumed that a substantial number of academically trained computer scientists ought to be familiar with the major themes of this topic. There is no question that this assumption would not meet universal approval in the larger community of computer professionals. However, it does seem reasonable that given the current level of concern about data security issues a formal introduction to the subject ought to be available at a representative set of universities. Additionally, it is at best mildly redundant to argue the merits of this particular proposition in a forum such as this one.. Thus, assuming that putative computer scientists ought to be exposed to the topic of data security as a formal requirement this presentation has two aims. The first is to assess the current level of instruction in data security as it is reflected in the published curricula of undergraduate departments of computer science. The second is to suggest how the present curriculum should be amended or expanded to include this topic.

It is possible to argue that data security will not be a truly important area of focus until it reaches the undergraduate curriculum. While this must surely sound presumptuous when stated by an academic, there are serious reasons for believing this to be the case. The basis for this assertion lies in the observation that to an increasingly large degree computer professionals are currently obtained from the ranks of individuals trained in university departments of computer science. Clearly, one need not regress too many years to find a time when this was not true. Indeed, a few more years backward in time reaches a point where there were no university departments of computer science at all. But that in not true now. University departments of computer science exist. They graduate increasingly large numbers of students. These students , their training and attitudes define in a real sense much of the current state of this discipline. Any topic ignored when training these student has an uncertain future as an area of major focus in the profession.

In a very real sense the larger community of computer science professionals is a victim of its own success. The explosive growth in employment and the parallel expansion of academic programs have been treated as the mixed blessing they indeed were. The profession has organized with surprising agility. Informal but apparently widely shared standards of training have evolved in a relatively short time. While there has been no formal structure for the imposition of these standards across higher education, the extent to which these informal standards have become the implicit norm is both encouraging and curiously disturbing. That some level of standardization of the curriculum has been achieved is encouraging. What is disturbing is that this standardization has overtones of the kind of rigidity that makes even incremental change in curricula virtually impossible in the more traditional disciplines.

An examination of the curriculum of university departments of computer science shows that a substantial majority of departments have accepted the suggestions of the various informal national committees. The courses, the content of the courses, the sequence in which the courses are taken and the related training in supporting scientific and mathematical topics is surprisingly uniform. The benefits of this uniformity are obvious.. It demonstrates that there is a valid core of identifiable knowledge that is computer science. It further suggests that academic departments of computer science are actively attempting to conform to objective standards in the design and implementation of courses of instruction. But there is a concurrent danger in any such uniformity. A discipline as relatively new and dynamic as computer science is not well served when the standards are established too early and especially when they become too rigid. The underlying discipline changes far too rapidly to allow the formulation of a canonical curriculum.

2. The Study

This study of information security in undergraduate education grew out of an attempt to design an advanced undergraduate/graduate survey course in data security. In designing any such course it seems reasonable to discover the current state of university course offerings in the given area. A brief survey of university catalogs suggested that there were virtually no current examples of course offerings at the level envisioned. This initial investigation led to a further more comprehensive study based on available public information on curriculum and course content.

In the study the published curriculum/course offering of 102 university catalogues were surveyed. The curriculum was examined to see what if any courses relevant to data security were taught and whether they were available to undergraduates. The prerequisites for those courses offered were noted. The catalogues were systematically searched for any department offering a course in the general area of data security. While the majority of such offerings were found in departments of computer science, some applicable course were found in departments of mathematics and in business related disciplines.

While any such survey has subjective elements, the defined criterion was to include any course offering that could by a knowledgeable reader be construed as treating in major part data/information security issues. Those courses that seemed to briefly touch on such issues were noted but not included.

The reason for carrying out the survey in the manner described are as follows. There are two major reasons for using only public documents. First in conducting a survey that demands a response, it is virtually impossible to obtain anything like the degree of compliance needed to insure results that are comprehensive. The attitude of many departmental administrators to surveys is negative at best and actively hostile at worst. This is especially true if the survey has no "official" standing in the sense of being actively sponsored by a national professional organization. A second reason is that voluntary responses to surveys of this type tend to be extremely selective. In this particular instance one would expect a strong bias in response from those institutions that do offer courses in data security. The purpose of the study is to determine one dimension of education in data security over all members of a specific set of universities. For these reasons it is best that the source of data in the study not be subject to the voluntary compliance of those institutions surveyed.

As a source for detailed information on course content the general university catalogue is not without its limitations. Course descriptions tend to be telegraphic at best. Not infrequently it is virtually impossible to divine the actual course content from the public description. However, in assessing the coverage of data security in undergraduate education, the public description of curriculum is probably the most accurate indicator of actual current attitudes. Thus, if an institution teaches a security course under the title of data base management or systems analysis or discrete mathematics(in the case of cryptanalysis) it very strongly suggests that the actual topic of the offering is somehow less than legitimate. If further, the public description of the offering is so obscured as to render it invisible to a knowledgeable reader, it is effectively as if the course does not in fact exist.

The 102 institutions surveyed include all 58 institutions listed in the 1982 National Commission survey of Graduate Programs In Computer Science.[1] The coverage of these 58 major graduate institutions is critical to the main conclusion of the survey. For any given year there are a variety of estimates as to exactly how many university graduates claim to major in computer science and exactly how many accredited colleges and universities offer a baccalaureate degree in computer science. Virtually all such information is the result of voluntary response to surveys and for that reason is at best stochastic. What is far more certain is the fact that national standards for computer science education will continue to be established by the policies of these leading institutions. It is granted that other institutions may offer equally good if not superior training and in a variety of ways be more innovative than the major graduate schools. But the fact remains that if these institutions in substantial numbers ignore a topic it is clearly not yet a part of the accepted undergraduate curriculum.

3. Results of the Study

The results of the survey may be summarized as follows. Of the 102 institutions in the survey 26 offered one or more courses on data security. Among the 26 institutions offering courses, 21 offer just one course and 5 offer two courses for a total of 31 course offerings. In these 31 courses, 25 are given by departments of computer science, two by departments of mathematics and one each by departments of management, business, accounting, and administrative science. Of the 31 courses offered 22 were judged to be available to advanced undergraduates by a generous interpretation of the course description. In those cases where the course was offered as a portion of the undergraduate curriculum in computer science the status of the course as to required, suggested, and optional was investigated. In no case was a security course required. in three cases the security course was formally suggested as an option

It is interesting that of the 31 data security courses offered 9 are available only to graduate students and 22 are listed as graduate courses.. Based on the admittedly brief descriptions available it was judged that only five of the 9 restricted courses had content that would have rendered them in actuality unavailable for almost any third or fourth year computer science undergraduate. Four of the 9 seemed to have content that was quite similar to that of survey courses offered at the undergraduate level.

Nine of the 31 courses on data security require no prerequisite course work. Six of the courses require a course on data structures and five a course on operating systems as a prerequisite.. Four require a course on analysis of algorithms. Five of the courses demand one or more mathematics courses as a prerequisite. Two require linear algebra, two require discrete mathematics, one requires applied algebra and one a course in mathematical analysis not further identified. Three of the courses have introductory statistics as a prerequisite Two of the courses listed as prerequisite one or more courses taught in departments other than computer science, mathematics and statistics. However from catalogue descriptions the content of these prerequisites seem to be essentially similar in subject matter to standard computer science courses . In all but three cases the courses listed as prerequisite are part of the undergraduate curriculum requirements for all computer science majors.

The geographic distribution of the course offerings is rather interesting. Fourteen of the the 31 courses offered are to be found at universities clustered in just two major metropolitan areas, San Francisco and Washington D.C. I suspect that this says something about demand and perhaps something about trends in the education of computer scientists.

The reason for offering relatively elementary, in content, security classes at the graduate level may only be surmised. There is, naturally, the issue of enrollment. Frequently classes will not be taught if some standard level of enrollment is not met. These standards are always more liberal in the case of graduate classes. It is surely the case that in many institutions the introduction of new courses at the undergraduate level is difficult. There are often a variety of bureaucratic hurdles and funding constraints that must be addressed. It is often easier to solve these difficulties for a graduate class than or an undergraduate one. Also, the introduction of an undergraduate class usually demands a modification of the existing undergraduate curriculum. This impinges on territorial imperatives within the department. In addition there are those who feel that while security issues are worth discussing they are not central to the education of computer scientists. It is the notion of imprinting. The fear is that if students meet a topic too early in their education they may tend to attach to it an importance it does not deserve. In this case the implication is that it is safe to offer security classes to graduate students since they have sufficient maturity to assess the relative importance of topics.

4. Recommendations

It seems evident that a substantial majority of current university graduates in computer science have no formal introduction to the issues of information security as a result of their university training. As stated earlier, it is axiomatic for the purposes of this discussion that such a condition is to be deplored. It should, in passing, be noted that the rapid standardization of the undergraduate curriculum in computer science will increasingly lead to a form of imprinting that will actively mitigate against attracting bright young scientists to this field. The reason for this is that as computer science education becomes more uniform among the universities, the student properly assumes that he will be taught those and only those topics that form the core of computer science. Any subject not included in this initial imprinting, is perceived as being unimportant.. It requires active and sustained indoctrination to convince him otherwise. The example of medical education at the turn of the century is instructive in the regard. As the accreditation of medical schools moved apace, the curriculum of the schools became quite standard. In most respects this was a valuable improvement in the quality of medical education. The schools graduated a standardized product that was predictably exportable nationwide. However, subjects, such as nutrition, that were not of major concern at the time the curriculum was standardized became unimportant. They remained unimportant long after serious scientific investigation demonstrated their role in the prevention and treatment of disease.

In the case of data security it is not too late to address the relative paucity of exposure of computer science majors. It is not difficult to make a case for the relative importance of this topic relative to many others currently available at the undergraduate level in almost all computer science curricula. That case should be made whenever the opportunity arises.

Those members of the general university community interested in data security should begin to offer courses on the subject. It would be best if these courses are offered at a relatively low level and are available to all undergraduate computer science majors with advanced standing. Some effort should be expended to see that these courses are included as viable options in the undergraduate curriculum. Following the survey course, upper level courses in systems security and cryptanalysis should be offered. These offerings need to be structured in such a way as to be available to both graduate students and advanced undergraduates. It is unwise to attach long lists of prerequisites to such courses.

While it is unlikely that a every institution would develop a variety of courses in security, it is important that some institutions do It establishes and helps to maintain the credibility of the subject and provides a nucleus of students interested in security topics. The most favorable interpretation of the survey seems to suggest that at present there are at best only two or three such universities in the entire nation.

The increasing importance of information security suggests that some coverage of the topic should be included in the standard curriculum at a relatively early level. It is unlikely that another required topic could be appended to the current list of essentials. The subject could,however,be included in anyone of a number of required courses such as those on on systems, human factors, discrete mathematics, etc. To have this accepted as a standard portion of the undergraduate curriculum will clearly demand the organized effort of those who feel that it belongs there.

Those individuals in industry and government who would like to see rather more exposure to data security in formal university education should take occasion to so state. To a far greater extent than more traditional disciplines, computer science curricula are market driven. This is especially true for the less prestigious instiutions. If it becomes clear that there is a market for basic education in data security a variety of suppliers will arise to fill that demand. In this regard it is important to note that the existence of curriculum offerings is as important as any specific training available from those offerings. The offerings legitimize the topic which in turn convinces even those who do not take the courses that the topic of data security is indeed worthy of serious attention.

References

[1] D.C. Rung, "Newest Rankings of Graduate Programs in Mathematics," Notices of the American Mathematical Society, Vol.30,Number 3, pp. 257-270, April 1983

COMPUTER SECURITY EDUCATION, TRAINING, AND AWARENESS: TURNING A PHILOSOPHICAL ORIENTATION INTO PRACTICAL REALITY

W. V. Maconachy, Ph.D.

U.S. Department of Defense
Fort George G. Meade, Maryland

Abstract

This paper discusses the scope of computer security education, and presents a schema for differentiating among Education, Training and Awareness activities.

Introduction and Overview

The mandate has been set down by Congress for the heads of federal agencies to design and develop computer security awareness, training and education programs for employees. In signing Public Law 100-235, "Computer Security Act of 1987", President Reagan set into motion requirements for the protection of sensitive unclassified information in federal computer systems. One of those requirements is the mandatory periodic training of all persons involved in the management, use or operation of federal computer systems. One and one-half years later, what has been accomplished? A report has been issued in the Fourth Annual Assessment of the National Telecommunications and Information Systems Security Committee. In that report the Committee states:

> To ensure that all government employees and contractors are aware of the INFOSEC considerations inherent in their duties and responsibilities, departments and agencies must continue to expand INFOSEC education and training programs. The President's National Security Advisor applauded the progress being made in enhancing the COMPUSEC posture of the nation. However, a more concerted effort in promoting security awareness throughout the government and private industry is required.

One problem associated with responding to the Committee's challenge is the lack of definitive differences among education, training and awareness activities. The schema outlined below provides a frame of reference for defining and building computer security programs on all three levels, and is applicable in both government and industrial environments, because the learning programs established in response to PL 100-235 are oriented towards protecting sensitive unclassified information. The noted criminologist Dr. Sherizen lists certain categories of information as sensitive enough to warrant protection (Figure 1). These categories exist in both government and industry.

Long Range or Contingency Plans
Major New Ventures
Acquisition or Sale of Business Assets
Major Planned Curtailment of Operations
Business Strategy or Product Technology
Future Product Design or Developments
Customer Lists
Accounting Records
Competitive Assessments and Comparisons
Travel Plans of Top Executives
Personnel Records
Financial Arrangements with Suppliers

Figure 1: Particularly Important Information Which Requires Protection

Building and Education, Training and Awareness Program

With that background covered, let's proceed to how employee sensitivity to the need for security in a government-industrial environment can be built. Before any awareness, training or educational activities can be pursued, an agency level goal should be developed and agreed upon. I suggest that a generic goal would be to develop in each employee an awareness for the need to make information security an integral part of his/her workday habits, and motivate the employees to develop the skills necessary to do so. These habits of behavior should encompass all aspects of security.

The all important first step in any campaign to increase the use of security practices is to obtain a commitment from the very highest corporate levels. Ultimately, an information security goal should be part of the written corporate philosophy. As part of that philosophy a clear definition and guideline must be provided which employees can use in determining which types of unclassified information a particular organization deems sensitive. Since PL 100-235 relegated this determination to each Federal agency, there will be a wide range of interpretations. This range of interpretation may influence the extent of education required under PL 100-235, but it does not diminish the need for such programs.

Once a corporate level commitment is obtained, the next logical step in the campaign is to understand and differentiate between security awareness, training and education. This is critical, from a practical standpoint, because security is always a hard sell. After all, where is the return on investment? If managers do not differentiate between security awareness, training and education programs early on in the campaign for excellence in security behaviors, funding may very likely be eaten away by education courses taken by employees in the name of increased awareness, and security educators will have little left to reach the greater number of employees not privy to such focused and individualized educational opportunities.

I offer the following continuum as a model for use in reaching that differentiation. Yes, the middle ground is gray, but the r del does serve several purposes:

A. As a point of departure
B. As a philosophical framework for operations
C. As a potential arbiter of bureaucratic lines (The training Dept. vs. the security Dept. responsibilities)
AND
D. As a tool for planning awareness activities appropriate for differing levels of thinking and learning.

LEARNING CONTINUUM

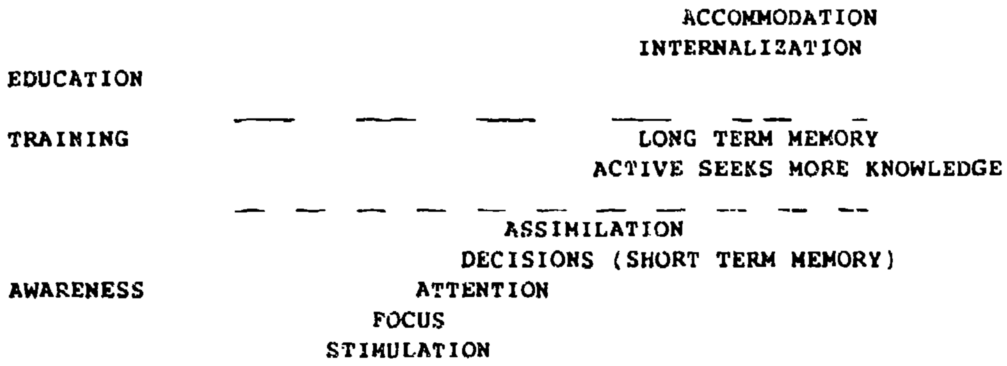


Figure 2: The learning Continuum

Awareness

Fundamental in this concept is the appreciation for the unique attention-getting/stimulation aspects of an awareness program versus the informational nature of education/training programs. Where awareness relies on reaching broad audiences with attractive packaging techniques, education and training programs are generally more focused in nature and typically restrict themselves to the so called, "sound educational methodologies".

The model presented here is based on a psychoneuronal model of learning. A model already proven useful in planning individual as well as a corporate stream of security consciousness.

Stimulation is the very first phase in learning. At this level some event triggers a basal level response that "wakes up" the individual's nervous system. In many work places, placing a security violation notice on the boss's desk manages to get him/her stimulated posthaste. BUT THAT IS NEGATIVE STIMULATION. Positive stimulation is preferred! This is achieved by a variety of techniques such as distributing an announcement of cash awards for security suggestions. The key is to use a color paper, or a style of announcement that is unique only to announcing monetary awards for security suggestions. One of the most common examples of focus is the use of different color badges to indicate specific levels of security clearance. Another common example is the use of different color paper, while maintaining the same shape and design for security information products which may change on a yearly basis.

Last year many agencies were using COMPUSEC information cards which were blue. When the content of the cards was updated, the cards were deliberately changed to a bright yellow so that users, who had these cards by their computers would know just by color if their cards were current. The idea behind this form of motivation is that seeing a specific shape or color, or hearing a particular tone will trigger senses to tune into the next stage of awareness; focus.

Obtaining learner Focus is a concept that is not so foreign to most of us. Imagine if you were going to go into your computer files looking for a specific item, and noticed a new file that read "REBENSHRACK". It probably would not take very long to recognize a nonsense word which is not part of your usual save file routine. That process is focus. In security awareness, focus can be obtained by a variety of techniques:

1. Having all personal computer screens come up with a security reminder when first turned on.
2. Changing the lock combinations on safes.
3. Issuing periodic security flyers with pictures of the CEO or agency head as a header to an article he wrote or endorsed.

Attention: The problem with focus is that humans tend to practice a tuning out process called acclimation. If a stimulus, once a powerful attention getter, is used repeatedly in the same environment the learner will selectively tune out the stimulus. The classic example of this is when a menu screen from an on-line search service changes patterns. What the reader has become accustomed to and conditioned to respond by simply ignoring and hitting a return key has changed. That change gets the user's attention. In INFOSEC awareness, the principle applied to this concept is to change bulletin boards, posters, and personal computer security messages and routines frequently.

Attention can also be obtained by using such GIMMICKS as key chains, magnetic tags, and other visual clues that offer daily reminders that security is a work habit.

Decisions: The first three steps, outlined above, usually take place in the human brain in a nano second. Once the learner's attention is attained, the leap to conscious decision becomes a critical yet most important part in changing employee behavior. The security world abounds in examples of primary decision making behavior (often termed the exercise of short term memory). Two key control operations, use of personal passwords and inserting employee card with PIN numbers are examples of primary decision making behavior. The purpose of imposing this level of effort on an employee is to make him/her think about what he or she is about to do.

On a higher plain, forcing the employee to exercise short term memory is necessary to evoke higher level security practices. These practices include:

1. Stopping to read a bulletin board or scrolling electronic message.
2. Deciding to read a new security regulation
3. Deciding to read the security corner of the company newsletter
4. Deciding to attend a security lecture.

Messages developed for employees at this level are often the most difficult to construct, yet are the key to leading an entire organization into a better security performance profile.

Assimilation: I have borrowed this term from the learning theorist Jean Piaget. It is a transformational component of learning through which all knowledge is acquired. It is a cognitive process in which an individual incorporates new experiences into already existing schema ofoperation. At this level of operation, the learner/employee consciously decides to incorporate security practices into his or her behavior. This experience is characterized by a growth in behavior pattern often without significant qualitative change in cognitive processing. Examples of how this behavior might be facilitated include activities such as:

1. Supplying employees who use personal computers with key rings that say, "lock me out when you go on break".
2. Offering security seminars that stimulate thoughts.
3. Offering security surveys, demonstrations or presentations to employees in their work environments.
4. Providing security-oriented video tapes.

Awareness vs. Training: The Gray Zone

There exists a gray zone between Awareness and training (as depicted in Figure No. 2). A gross distinction between awareness and training is that in awareness activities the learner is a passive recipient of information, while in the training environment the learner has a more active role in the learning process. A primary role of awareness programs is to motivate employees/learners to move into a training mode and and actively seek more knowledge. A fundamental goal of training programs is to motivate learners to move knowledge and skills from short term memory into long term memory. Very often these knowledges and skills become chained sequences of behavior which require little higher level mental processing.

In agencies where these functions are divested, collaboration between the corporate providers of training and the corporate planners of INFOSEC awareness is essential to developing and delivering quality learning experiences. Activities in this domain include:

1. Advertising education programs available through such agencies as DIS, DoD, OPM, Private consultants, and colleges and universities.
2. Sponsoring training seminars
3. Planning and executing an annual security week.

Here, the awareness plan includes:

a. Table tents on cafeteria tables
b. Announcements on Electronic bulletin boards
c. Announcements on Corridor bulletin boards
d. Flyers,

And, the training plan includes:

a. Formal hands-on seminars
b. On-site short courses and briefings

Training vs. Education

The debate over differences between training and education has raged since time immortal. I offer the following point of departure. Where awareness relies on reaching a broad audience and the use of attractive packaging techniques, training and education programs are generally more focused in nature and typically restrict themselves to the so called, "Sound Educational Methodologies". The distinction between training and education can be made by examining the intent and scope of instruction. In a training environment the employee is taught to use specific skills as part of exacting job performance. In an educational context the employee would be encouraged to examine and evaluate not only skills and methods of work but fundamental operating principles and tenants upon which job skills are based. The employee/learner is using internalized concepts and skills to perform operations such as analysis, evaluation and judgement to reach higher cognitive level decisions which lead to the <u>accommodation</u> of newly integrated knowledge and skill. In the context of this paper, accommodation is an end process in which the learner makes a conscious decision to modify existing ways of thinking and responding in order to satisfy new experiences and knowledge. Very often, accommodation results in significant qualitative changes in performance. An example of operations at this level would be designers of networks which require interpretive techniques to assure varying levels of security. Capability to operate at this level is fostered through educational programs and processes.

Figure 3 provides an example of computer security content which is based on the learning continuum principle. Implicit in the example is the dynamic interrelation and interdependence of awareness, training and education activities.

Goal: Facilitate the increased use of password protection among all employees.

Awareness Activity: Reminder Stickers for keyboards

Training Activity: Computer Based Instruction on the use of passwords for agency-specific machines.

Education Activity: Recognized COMPUSEC expert provides employees opportunity to explore why passwords are used in general, and evaluate the current agency protection techniques.

Figure 3: An Example

<u>Summary</u>

A true computer security learning program incorporates concepts and elements from each level, and presents the employee/learner with a totally integrated succession of experiences. Figure No. 4 summarizes activities which may be found on each level of operation. It is by no means inclusive.

AWARENESS

Stimulation:
- Security-only colors
- Security-only music theme

FOCUS:
- Change Locks
- Reminders

ATTENTION:
- Bulletin Boards
- Flyers
- Posters

DECISIONS:
- Read Security Regulations
- Read magazines
- Attend Lecture

ASSIMILATION:
- Key ring with messages
- Short seminars
- Short demonstrations
- Video tape programs

TRAINING

ACTIVE KNOWLEDGE SEEKER
- Self-Paced course
- OJT
- Conferences

LONG TERM MEMORY
- Computer-Based instruction
- Multi-session seminar

EDUCATION

INTERNALIZATION
- Point Papers
- Study groups

ACCOMMODATION
- Long term training
- Research and deliver briefing

Figure 4: Activities per Level

This paper offers some ideas and an approach to consider in building information systems security practices into COMPUSEC awareness, training, and education programs. Your imaginations can expand the opportunities and experiences which can reach your employees. However, do not let this analytical view of the awareness/training/education continuum, cloud your view for the need for a truly integrated program. A host of activities should be carefully constructed so as to provide employees at all levels a total program of systems security. This integrated approach requires the melding of many talents and coalescence amongst often separate groups.

Turning hypothetical construct into realty is hard work But it can be an exciting challenge. A challenge, if unmet may result in utter calamity at:

The personal level
The corporate level
The national Level

The challenge provided to government personnel is to take up the task of developing and implementing a well orchestrated government-wide information systems security awareness, training and education model. A model which may begin to unfold through the development of computer security awareness, training and education programs. The challenge requires great vision for the future, and cannot be dismayed by often harsh realities of budget, lowered priorities or apathy.

ALTERNATE PAPERS

A LEAST FIXED POINT APPROACH TO INTER-PROCEDURAL INFORMATION FLOW CONTROL

Masaaki Mizuno
Department of Computing and Information Sciences
Kansas State University
Manhattan, Kansas 66506
masaaki@ksuvax1.cis.ksu.edu

1. Introduction

Information flow control regulates the flow of information between classified objects [7]. Given a set of "security classes" corresponding to the sensitivities of information and a specification of all the paths among objects by which information is allowed to flow (an information flow policy), an information flow mechanism must guarantee that the flows caused by program executions do not violate the specification. Denning introduced the use of a complete lattice structure to define an information flow policy [5]

Based on a policy defined by a complete lattice, Denning developed a compile-time algorithm for certifying the secure execution of a program in an environment in which the security class of each object (program variable or file) remains constant throughout the lifetime of the program [6]. In this environment, a programmer needs to specify the security class of every one of the program variables. Since constant security classes of parameters must also be specified, separate versions of functionally equivalent procedures are required to handle different security classes of parameters. This is a major drawback of this approach.

Andrews and Reitman developed a compile-time certification technique based on Hoare's program verification [3]. This mechanism allows the security class of each variable either to remain constant or to change during execution of the program. The verification of a procedure invocation requires previous verification of the body of the called procedure and previous establishment of the pre/post-conditions (of the called procedure). Thus, the verification of procedure PROC requires previous verification of all the procedures which are potentially invoked by PROC.

In earlier work, we presented an information flow certification mechanism designed for distributed object-oriented systems [8]. The mechanism has the following features:

1. The security classes of object variables must remain constant. The security classes of other program variables can either remain constant or change during execution of a program.

2. Each procedure can be compiled and its "internal" security established independent of other procedures.

The mechanism combines compile-time and run-time algorithms. The compile-time algorithm establishes the internal information flow security of an individual procedure and also creates a special data structure for efficient run-time certification. The run-time algorithm completes the certification of the entire program at message passing time by verifying information flows caused by procedure invocations.

This paper expands earlier work; it presents an information flow certification mechanism which combines compile-time, link-time and run-time algorithms. In this approach, a user can specify the security class of each variable either to remain constant or to change during execution. The compile time algorithm is basically the same as the one presented in [8][1]. The link-time algorithm uses the data structure generated by the compile-time algorithm and calculates potential information flows caused by parameter passing and global variable access. The run-time algorithm completes the certification by verifying the flows caused by external file access.

We assume that, as described in [1], a security label is associated with each storage object (e.g. external files[2]), and maintained by a TCB (Trusted Computing Base). Thus, as a reasonable assumption, the security classes of exte aal files remain constant during execution of a (user) program. The specific security classes of files accessed by a program, however, do not have to be specified until run time[3]. This eliminates a need for separate versions of functionally equivalent programs for different external files. Immediately prior to the execution of a program, the user specifies all the files accessed by the program (binding). The decision whether the program is allowed to execute is made at the binding time by the run-time algorithm (performed by the TCB).

2. A Definition of Flow Control

This section presents basic definitions of information flow control and an information flow policy. An information flow from variable 'x' to variable 'y', which is denoted by 'x ⇒ y', occurs if information in 'x' is transferred to 'y'. It indicates that information in 'y' could be used to derive information in 'x'.

Flows can be classified as explicit or implicit. An explicit flow from variables $a_1, \ldots, a_n$ to variable 'x' occurs when an execution directly assigns information derived from $a_1, \ldots, a_n$ to 'x'. An implicit flow from variables $a_1, \ldots, a_n$ to variable 'x' occurs when an execution of a statement which assigns some information to 'x' is conditioned upon values derived from $a_1, \ldots, a_n$. For example, the statement

if a > 0 **then** x := y **else** y := z

causes an explicit flow from 'y' to 'x' only when a > 0, and from 'z' to 'y' only when a ≤ 0. The statement also causes an implicit flow from 'a' to both 'x' and 'y' regardless of the value of 'a'.

The underlying theory of information flow control is based on the complete lattice (SC, ≤, ⊕, ⊗) [9] introduced by Denning [5], where

1. SC is a finite set of security classes;

2. ≤ is a binary relation which induces a partial ordering on the security classes in SC;

3. ⊕ is an associative and commutative binary operator on SC, denoting the least upper bound, e.g. A ⊕ B is the least upper bound of security classes A and B;

[1] In order to optimally adjust to the link-time algorithm, the data structure generated by the compile time-algorithm is slightly different from the one described in [8]. However, the concept is the same.

[2] We assume that the term "external files" includes I/O devices.

[3] For simplicity, we assume that if the security classes of program variables other than external files are specified to be constant, they must be defined at compile time.

4. $\otimes$ is an associative and commutative binary operator on SC, denoting the greatest lower bound, e.g. A $\otimes$ B is the greatest lower bound of security classes A and B;

5. SC has the greatest lower bound LOW and the least upper bound HIGH such that LOW $\leq$ A and A $\leq$ HIGH for all A in SC.

Information of class A is allowed to flow into an object in class B if and only if A $\leq$ B is implied by the lattice. For simplicity, the examples in this paper assume a linear lattice of security classes consisting of UNCLASSIFIED (= LOW), CONFIDENTIAL, SECRET, TOPSECRET (= HIGH).

A progra i variable may be either statically or dynamically bound to a security class. A "statically bound variable" is assigned a fixed security class at the time of its definition. The security class of a "dynamically bound variable" changes with the class of its associated information. For notational convenience, if 'x' is a variable, then the security class of 'x' will be denoted by '$\underline{x}$'.

If 'y' is a statically bound variable, then the flow 'x $\Rightarrow$ y' is secure if and only if the relation '$\underline{x} \leq \underline{y}$' is implied by the lattice. Otherwise, a security violation occurs. Note that if 'y' is a dynamically bound variable, '$\underline{y}$' becomes '$\underline{x}$' and no security violation occurs.

3. Overview of our Information Flow Control Mechanism

Our mechanism consists of three components: a compile-time algorithm, a link-time algorithm and a run-time algorithm. The compile-time algorithm partially certifies the security of each procedure independent of other procedures. It also generates symbolic equations representing the security classes of global variables and parameters.

The link-time algorithm calculates the least fixed points of the equations generated by the compile-time algorithm to determine inter-procedural information flows caused by global variable access and parameter passing.[4] The link-time algorithm also generates a set of equations representing information flows caused by external file access. We assume that these equations are stored securely within the TCB.

At run-time, when all the external files accessed by the program are specified (immediately prior to the execution of the program), the TCB certifies the security of potential information flows to all the statically bound variables. This certification is performed based on the sensitivity labels of the files and the equations generated by the link-time algorithm. If all the flows are certified to be secure, the execution is allowed to begin; otherwise the execution is denied.

We assume the following syntax for a procedure declaration statement:

$$\textbf{procedure}\ \mathrm{PROC}\ (\textbf{IN}\ x_1, \ldots, x_l;\ \textbf{OUT}\ y_1, \ldots, y_m)$$

where the **IN** parameters are "call by value" and the **OUT** parameters are "call by result". The mechanism does not handle other types of parameter passing mechanisms.

We first identify all possible input and output values to/from a procedure. We define the terms "input variables" and "output variables" to stand for variables which carry input values to the procedure and output values from the procedure, respectively.

Input variables of a procedure PROC are:

(1) actual **IN** parameters of PROC

[4] We assume the scope of global variables (except for external files) to be within a program.

(2) the global variables (including external files) read by PROC

(3) formal **OUT** parameters returned from procedures that are called by PROC.

Output variables of PROC are:

(4) actual **OUT** parameters of PROC

(5) the global variables (including external files) written by PROC

(6) formal **IN** parameters to procedures that are called by PROC.

The compile-time algorithm constructs equations that express the potential run-time inter-procedural information flows in symbolic form. In order to do this, a "symbolic class expression" is generated for each variable in terms of the classes of the input variables (1)-(3). A symbolic class expression represents the class of information in terms of the classes of variables from which it is composed. For example, the class of information in the expression "A + B * C − D / E" is symbolically denoted by "$\underline{A} \oplus \underline{B} \oplus \underline{C} \oplus \underline{D} \oplus \underline{E}$".

If input variables (1) (2) and (3) are dynamically bound, their security classes cannot be determined at compile time. Even though external files are statically bound, their security classes may not be determined until run time. During compilation, the classes of these variables are established as "security variables". Security variables are symbolically denoted by

1. $\underline{\text{parameter-name}}$ (for formal **IN** parameters of the procedure being compiled),

2. $\underline{\text{procedure-name.variable-name}}$ (for actual **OUT** parameters of procedures),

3. $\underline{\text{variable-name}}$ (for dynamically bound global variables), or

4. $\underline{\text{file-name}}$ (for external files).

For example, if the procedure being compiled is F(**IN** a, b), then the classes of dynamically bound parameters 'a' and 'b' are symbolically denoted by '$\underline{a}$' and '$\underline{b}$', respectively. If this procedure invokes a procedure G as G(**IN** x, **OUT** y, z), the classes of 'y' and 'z' are symbolically denoted by $\underline{\text{G.y}}$ and $\underline{\text{G.z}}$, respectively. Furthermore, if F(**IN** a, b) reads from both a terminal STDIN and a dynamically bound global variable GV, the security classes of STDIN and GV are denoted by $\underline{\text{STDIN}}$ and $\underline{\text{GV}}$, respectively.

Based on these symbolic class expressions, the compile-time algorithm generates a "symbolic class equation" for each output variable (4) (5) and (6), and each statically bound variable. The equation has the form

$$\underline{\text{variable}} = \text{“symbolic class expression”}$$

which states that the security class of information given by "symbolic class expression" flows into "variable."

At link time when symbolic class equations for all procedures in a program are collected, the link-time algorithm finds, for each security variable corresponding to a parameter and a dynamically bound global variable, a symbolic class expression which denotes the potential run-time security class of information flowing to the variable.

Dynamically bound global variables require special consideration. Consider the following program segment:

```
x := GV1; -- (a)
call PROC (...); -- (b)
y := GV1; -- (c),
```

where GV1 is a dynamically bound global variable. Even though both statements (a) and (c) refer to the same variable GV1, the security class of GV1 in (c) may be different from that in (a). This is because the procedure call to PROC in (b) may change the value and the security class of GV1. In general, a trace of the security classes of such a dynamically bound global variable may depend on the order of procedure invocations and cannot be determined at compile time or link time.

The compile-time algorithm simply uses the security variable whenever a global variable is referenced in a procedure. At link time, each such security variable is substituted by the least upper bound of all the symbolic class expressions for the global variable. In the above example, therefore, our certification analysis assumes that the same security class of values flows to both 'x' and 'y' (and all other variables in the program which refer to GV1). Since this approach considers the worst case, it is safe but may not be precise.

The security classes of external files are not determined until run-time. Since the security class of a variable may be dependent on the classes of external files, the link-time algorithm computes the security class of a variable to be a pair consisting of a fixed security class (in the security lattice) and the set of the security variables corresponding to files whose information will flow into the variable. Let the power set of a set of the security variables for files be P. Then P is a lattice, and '$\leq$', '$\oplus$' and '$\otimes$' on P are defined as

1. $a \leq b$ iff $a \subseteq b$,
2. $a \oplus b = a \cup b$, and
3. $a \otimes b = a \cap b$

where $a, b \in$ P.

Now let the security lattice of the system be S. Then S * P (the direct product of S and P) is also a lattice, and '$\leq$', '$\oplus$' and '$\otimes$' on S * P are defined as

1. $< a_1, b_1 > \quad \leq \quad < a_2, b_2 >$ iff $a_1 \leq a_2$ and $b_1 \leq b_2$
2. $< a_1, b_1 > \oplus < a_2, b_2 > = < a_1 \oplus a_2, b_1 \oplus b_2 >$
3. $< a_1, b_1 > \otimes < a_2, b_2 > = < a_1 \otimes a_2, b_1 \otimes b_2 >$

where $a_1, a_2 \in$ S, $b_1, b_2 \in$ P.

The link-time algorithm creates n equations with n unknown variables, where n is the number of symbolic class equations of a program. The domain of each variable in the equations is S * P. By solving the equations, the algorithm can determine the security classes (in the domain S * P) of information which would potentially flow into the statically bound variables in the program. These security classes are output for run-time certification.

At run-time, when all the files accessed by the program are specified, the TCB binds the sensitivity labels of the files to the associated security variables to determine any potential flow to each statically bound variable. The program is certified to be secure and can be executed only if the security classes of all such flows are less than or equal to the security classes of the associated statically bound variables.

4. The Compile-Time Algorithm

Our compile-time algorithm is based on Dennings's compile-time certification mechanism [6]. The details of the algorithm are shown in [8]. Since the focus of attention in this paper is the link-time/run-time algorithms, we only explain a special flow called "implicit inter-procedural information flow" and then show a simple example.

If a procedure call is conditioned upon some variable(s), then there is an implicit inter-procedural information flow. For example, in the following statement in procl

if a $\geq$ 0 **then** proc2(),

there is a flow from 'a' into proc2 (and those procedures called by proc2, etc.), that is, for every global and local variable 'z' in proc2 and subsequently called procedures, '$a \Rightarrow z$'. In order to handle this implicit flow, the compile-time algorithm constructs a special symbolic class expression "$\underline{\text{proc2.implicit}} = \underline{a} \oplus \cdots$." $\underline{\text{proc2.implicit}}$ represents the class of implicit inter-procedural flow into proc2 and is derived by

$$\underline{\text{proc2.implicit}} = \underline{SV_1} \oplus \cdots \oplus \underline{SV_n} \oplus \underline{\text{implicit}}$$

where SV_i denotes the i_{th} variable on which the invocation is locally conditioned, and $\underline{\text{implicit}}$ denotes the class for the implicit inter-procedural flow into procedure procl from its caller. $\underline{\text{implicit}}$ and $\underline{\text{proc2.implicit}}$ are treated in the same manner as formal **IN** parameters to procl and actual **OUT** parameters from procl to proc2, respectively. Note that an execution starts with the 'main' procedure; therefore, $\underline{\text{implicit}}$ for 'main' is LOW.[5]

In order to show how the compile-time algorithm works, we now present an example which consists of three procedures: 'main', 'f' and 'g'. The program is shown in Figure 1. Procedure 'main' calls 'f', and 'f' and 'g' call each other recursively. The program accesses external files FILE1 and FILE2, and I/O devices STDIN and STDOUT. Their security classes are determined at run time. Dynamically bound global variable 'GV1' is accessed by all three procedures. We will simulate the algorithm on procedure 'f'.

For local variable 'c' defined in line (a) in Figure 1, there are explicit flows from GV1 (whose security class is $\underline{\text{GV1}}$) and constant 2, and there is an implicit inter-procedural flow (whose security class is $\underline{\text{implicit}}$). Thus, the algorithm constructs the equation

$$\underline{c} = \text{LOW} \oplus \underline{\text{GV1}} \oplus \underline{\text{implicit}}.$$

For (b), since there are implicit flows from 0 and 'a' to **IN** parameter 'c' of invocation 'g' as well as implicit inter-procedural flow, the following equation is constructed:

$$\underline{\text{g.c}} = \underline{c} \oplus \underline{a} \oplus \text{LOW} \oplus \underline{\text{implicit}} = \text{LOW} \oplus \underline{a} \oplus \underline{\text{GV1}} \oplus \underline{\text{implicit}}$$

Since 'c' is an output variable of 'f', the algorithm outputs this equation. The algorithm also generates the following equation for the implicit inter-procedural flow for invocation 'g':

$$\underline{\text{g.implicit}} = \text{LOW} \oplus \underline{a} \oplus \underline{\text{implicit}}.$$

Variable 'b' is assigned a value in (c) and (d). Since the choice of the **then** branch or the **else** branch cannot be determined at compile time, the algorithm must consider both paths. There are implicit flows from 'a' and 0 as well as an implicit inter-procedural flow to 'b'. Also there is an explicit flow either from 'GV1' or 'FILE1' (whose security class is $\underline{\text{FILE1}}$) to 'b', the following equation is generated:

[5] With the assumption that execution of a program is not conditioned on any sensitive information.

$$\underline{b} = \text{LOW} \oplus \text{a} \oplus \underline{\text{GV1}} \oplus \underline{\text{FILE1}} \oplus \underline{\text{implicit}}.$$

Since 'b' is an output variable of 'f', the algorithm outputs the equation. The symbolic class equations for 'f' and those for main and 'g' are shown in Figure 2.

5. The Link-Time Algorithm

The link-time algorithm first examines the correspondence between formal and actual parameters, and the correspondence among global variables to find, for each security variable, the corresponding symbolic class expression. Since the link-time algorithm treats security variables representing files as constants, the term 'security variable' in the following paragraphs denotes a security variable corresponding to either a parameter or a dynamically bound global variable.

Consider a procedure P. The security variables appearing in the symbolic class equations of P correspond to either actual **IN** parameters of procedures which call P, or formal **OUT** parameters of procedures invoked by P, or incoming implicit inter-procedural flows from procedures that call P. Suppose P is called by procedures $R_1 \ldots R_n$ and suppose P calls another procedure Q. The symbolic class expressions corresponding to the formal **OUT** parameters of Q are found in the set of equations for Q.

The symbolic class expressions corresponding to the actual **IN** parameters or the outgoing implicit inter-procedural flows from the callers to P are found in the set of equations for $R_1 \ldots R_n$. For the same formal **IN** parameter in P, the algorithm concatenates (using $\oplus$) symbolic class expressions for the corresponding actual **IN** parameters found in equations for $R_1 \ldots R_n$ to form a single symbolic class equation. A similar procedure is also applied to form a single equation for the implicit inter-procedural flow.

As mentioned in Section 3, the algorithm then concatenates (using $\oplus$), for each dynamically bound global variable, all the symbolic class expressions corresponding to the same variable. Thereafter, there is exactly one symbolic class equation corresponding to each security variable.

The algorithm assigns a distinctive number to each symbolic class equation and renames the left side variable of each symbolic class equation with X_i where i is the number assigned to the equation. Based on the correspondence between formal and actual parameters, the link time algorithm replaces every security variable appearing in the right hand side of a symbolic class equation with the corresponding X_i. In the following discussion, we assume that $F_1, \ldots, F_m$ denote security variables corresponding to external files which appear in the right hand sides of equations.

Assume that the algorithm has created the following n equations with n unknown variables:

$$\begin{aligned} X_1 &= f_1(X_1, X_2, \ldots, X_n) \\ X_2 &= f_2(X_1, X_2, \ldots, X_n) \\ &\vdots \\ X_n &= f_n(X_1, X_2, \ldots, X_n). \end{aligned}$$

Define a n,n-place function

$$F = \lambda(X_1, \ldots, X_n)(f_1(X_1, X_2, \ldots, X_n), \ldots, f_n(X_1, X_2, \ldots, X_n)).$$

Let S be the security lattice of the system and P be a lattice constructed from the power set of $\{ F_1, \ldots, F_m \}$. The domain of each variable in the above equations is $S * P$. Thus, the domain of F is defined by $(S * P) * (S * P) \ldots * (S * P)$ $(= (S * P)^n$ — the direct product of n $(S * P)$s). Since both S and P are complete lattices, $(S * P)^n$ is a complete lattice, and '$\leq$', '$\oplus$' and '$\otimes$' on $(S * P)^n$ are defined as

1. $< x_{i1}, \ldots, x_{in} > \quad \leq \quad < x_{j1}, \ldots, x_{jn} >$ iff $x_{i1} \leq x_{j1}, \ldots, x_{in} \leq x_{jn}$
2. $< x_{i1}, \ldots, x_{in} > \oplus < x_{j1}, \ldots, x_{jn} > = < x_{i1} \oplus x_{j1}, \ldots, x_{in} \oplus x_{jn} >$
3. $< x_{i1}, \ldots, x_{in} > \otimes < x_{j1}, \ldots, x_{jn} > = < x_{i1} \otimes x_{j1}, \ldots, x_{in} \otimes x_{jn} >$

where $x_{ik}, x_{jk} \in (S * P), 1 \leq k \leq n$.

The following characteristics of F guarantee that F has a least fixed point [9]:

- Since the domain of F is finite, it is a complete lattice, which is a pointed complete partial ordering (cpo); and
- F uses only a $\oplus$ operator which is continuous in finite domain, thus F is continuous.

The least fixed point of F, fixF = $(Y_1, Y_2, \ldots, Y_n)$, is a minimal solution for the above set of equations.[6]

In order to find the least fixed point of F, a standard iterative algorithm shown in Figure 3 is used.[7] For each X_i, the algorithm finds a solution $< A_i, \{\underline{F_{i1}}, \ldots, \underline{F_{ij}}\} >$, where $A_i \in$ S and $\{\underline{F_{i1}}, \ldots, \underline{F_{ij}}\} \in$ P. Security variables $\underline{F_{i1}}, \ldots, \underline{F_{ij}}$ are later replaced with security classes in domain S. The least upper bound of A_i and these security classes is the security class of information of X_i. For each X_i which corresponds to the class of information flowing to a statically bound variable SV_i, the link-time algorithm outputs the following equation:

'security class of SV_i' $\geq A_i \oplus \underline{F_{i1}} \oplus \ldots \oplus \underline{F_{ij}}$.

Note that if SV_i is an external file whose class is specified at run time, its security class is represented by the corresponding security variable.

The TCB, at run-time, certifies each equation by replacing every security variable with the sensitivity label of the corresponding file.

6. An Example

In this section, we apply the link-time and run-time algorithms to the set of symbolic class equations shown in Figure 2. Treating security variables representing files as constants, the symbolic flow equations for 'main' are rewritten as follows:

$\underline{\text{f.implicit}}$ = (LOW, { }) -- (1)
$\underline{\text{f.a}}$ = (LOW, {$\underline{\text{STDIN}}$}) — (2)
$\underline{\text{GV1}}$ = (LOW, {$\underline{\text{STDIN}}$}) — (3)
$\underline{\text{STDOUT}}$ = (LOW, { }) $\oplus$ $\underline{\text{f.b}}$ -- (4).

[6] If fixF is the least fixed point of F, the equation "$(\lambda(X_1, \ldots, X_n)(f_1, \ldots, f_n)).(Y_1, \ldots, Y_n) = (Y_1, \ldots, Y_n)$" is satisfied. Thus, fixF is a solution of the above n equations.

[7] The algorithm is a modification of the one presented in [2].

The equations for procedures 'f' and 'g' are similarly rewritten.

For each security variable for a parameter, the corresponding symbolic class expression is found by examining the correspondence between formal and actual parameters. Since both 'main' and 'g' call 'f', the symbolic class equations corresponding to the security variable for the formal **IN** parameter 'a' of procedure 'f' are combined. The actual **IN** parameters 'a' of procedure 'main' and 'z' of procedure 'g' match the formal **IN** parameter 'a' of procedure 'f'. Since 2 and 9 are assigned to the symbolic class equations for 'a' of 'main' (f.a) and 'z' of 'g' (f.z), respectively, a in the equations for 'f' is replaced by $X_2 \oplus X_9$. This replacement is denoted by

- a (f) = f.a (main) $\oplus$ f.z (g) = $X_2 \oplus X_9$.

Similarly, the symbolic class equations corresponding to the security variable for the incoming implicit inter-procedural flow of 'f' are combined as follows:

- implicit (f) = f.implicit (main) $\oplus$ f.implicit (g) = $X_1 \oplus X_8$.

Based on other formal and actual parameters, the following substitutions are also made:

- f.b (main) = b (f) = X_5
- x (g) = g.c (f) = X_7
- implicit (g) = g.implicit (f) = X_6
- f.y (g) = b (f) = X_5.

Since 3 and 11 represent the flows to GV1, the following substitution for GV1 is made:

- GV1 (f) = $X_3 \oplus X_{11}$.

Based on the above observations, the following eleven equations with eleven unknown variables are constructed:

$$
\begin{aligned}
X_1 &= (\text{LOW}, \{\ \}) \\
X_2 &= (\text{LOW}, \{\underline{\text{STDIN}}\}) \\
X_3 &= (\text{LOW}, \{\underline{\text{STDIN}}\}) \\
X_4 &= (\text{LOW}, \{\ \}) \oplus X_5 \\
X_5 &= (\text{LOW}, \{\underline{\text{FILE1}}\}) \oplus X_1 \oplus X_2 \oplus X_3 \oplus X_8 \oplus X_9 \oplus X_{11} \\
X_6 &= (\text{LOW}, \{\ \}) \oplus X_1 \oplus X_2 \oplus X_8 \oplus X_9 \\
X_7 &= (\text{LOW}, \{\ \}) \oplus X_1 \oplus X_2 \oplus X_3 \oplus X_8 \oplus X_9 \oplus X_{11} \\
X_8 &= (\text{LOW}, \{\ \}) \oplus X_6 \oplus X_7 \\
X_9 &= (\text{LOW}, \{\ \}) \oplus X_6 \oplus X_7 \\
X_{10} &= (\text{LOW}, \{\ \}) \oplus X_5 \oplus X_6 \oplus X_7 \\
X_{11} &= (\text{LOW}, \{\underline{\text{FILE2}}\}) \oplus X_6 \oplus X_7.
\end{aligned}
$$

The iterative algorithm generates the following sequence:[8]

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
1	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$	$L\{\}$
2	$L\{\}$	$L\{S\}$	$L\{S\}$	$L\{\}$	$L\{S1\}$	$L\{S\}$	$L\{S\}$	$L\{S\}$	$L\{S\}$	$L\{S1\}$	$L\{S2\}$
3	$L\{\}$	$L\{S\}$	$L\{S\}$	$L\{S1\}$	$L\{S12\}$	$L\{S\}$	$L\{S2\}$	$L\{S2\}$	$L\{S2\}$	$L\{S12\}$	$L\{S2\}$
4	$L\{\}$	$L\{S\}$	$L\{S\}$	$L\{S12\}$	$L\{S12\}$	$L\{S2\}$	$L\{S2\}$	$L\{S2\}$	$L\{S2\}$	$L\{S12\}$	$L\{S2\}$
5	$L\{\}$	$L\{S\}$	$L\{S\}$	$L\{S12\}$	$L\{S12\}$	$L\{S2\}$	$L\{S2\}$	$L\{S2\}$	$L\{S2\}$	$L\{S12\}$	$L\{S2\}$.

[8] L, S, 1 and 2 stand for LOW, STDIN, FILE1 and FILE2, respectively.

By using the values for X_4 and X_{10}, the link-time algorithm generates the following two equations:

$$\begin{aligned}&\underline{\text{STDOUT}} \geq \text{LOW} \oplus \underline{\text{STDIN}} \oplus \underline{\text{FILE1}} \oplus \underline{\text{FILE2}}\\ &\underline{\text{FILE2}} \geq \text{LOW} \oplus \underline{\text{STDIN}} \oplus \underline{\text{FILE1}} \oplus \underline{\text{FILE2}}.\end{aligned}$$

Assume that at run time a user with security clearance CONFIDENTIAL logs onto the system. The TCB authenticates the user's identity and determines his clearance. We assume that the security class of his terminal (STDIN and STDOUT) is bound to CONFIDENTIAL at this point. First, consider that when he issues a command to execute the program, he binds files with security label CONFIDENTIAL to FILE1 and FILE2. The TCB replaces $\underline{\text{STDIN}}$, $\underline{\text{STDOUT}}$, $\underline{\text{FILE1}}$ and $\underline{\text{FILE2}}$ in the above equations with CONFIDENTIAL. The resulting equations become

$$\begin{aligned}&\underline{\text{STDOUT}} = \text{CONFIDENTIAL} \geq \text{CONFIDENTIAL}\\ &\underline{\text{FILE2}} = \text{CONFIDENTIAL} \geq \text{CONFIDENTIAL}.\end{aligned}$$

Since potential flows to both statically bound variables are secure, the TCB allows the execution.

Next, assume that the user binds a SECRET file to FILE1 and an UNCLASSIFIED (= LOW) file to FILE2. The TCB reduces the equations to the following:

$$\begin{aligned}&\underline{\text{STDOUT}} = \text{CONFIDENTIAL} \not\geq \text{SECRET}\\ &\underline{\text{FILE2}} = \text{UNCLASSIFIED} \not\geq \text{SECRET}.\end{aligned}$$

Since both flows are potentially insecure, the TCB denies the execution.

7. Conclusion

This paper has described an information flow certification mechanism which combines compile-time, link-time and run-time algorithms. The compile-time algorithm is the same as the one we earlier developed for distributed object-oriented systems [8]. The link-time algorithm determines inter-procedural information flows caused by parameter passing and global variable access. The algorithm does this by calculating the least fixed points of the equations generated by the compile-time algorithm. The run-time algorithm is described in the context of the TCB. It completes the certification by binding the security classes of external files to the equations generated by the link-time algorithm.

The mechanism has the following features:

1. Program variables can be either statically bound or dynamically bound to security classes. If external files are statically bound, their security classes do not have to be determined until run time. This feature eliminates a need for separate versions of functionally equivalent programs for different security classes of variables.

2. Each procedure can be compiled and its "internal" security established totally independent of other procedures.

We are currently working on the mathematical foundations of our information flow analysis within Cousot and Cousot's "abstract interpretation" method [4]. This will be a basis for certification of a compiler, linker, and TCB system.

References

[1] *Department of Defense Trusted Computer System Evaluation Criteria.* DoD, dod 5200.28-std edition, December 1985.

[2] A. V. Aho and J. D. Ullman. *Principles of Compiler Design.* Addison Wesley, 1979.

[3] G. R. Andrews and R. P. Reitman. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56-76, 1980.

[4] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principle of Programming Languages*, pages 238-252, 1977.

[5] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236-243, 1976.

[6] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504-512, 1977.

[7] C. E. Landwehr. Formal models for computer security. *Computing Surveys*, 13(3):247-278, 1981.

[8] M. Mizuno and A. E. Oldehoeft. Information flow control in a distributed object-oriented system with statically bound object variables. In *Proceedings of the 10th National Computer Security Conference*, pages 56-67, 1987.

[9] D. A. Schmidt. *Denotational Semantics — A Methodology for Language Development.* W. C. Brown Publisher, 1986.

```
external files
    STDIN : keyboard of integer;
    STDOUT : display of integer;
    FILE1 : file of integer;
    FILE2 : file of integer;
global var
    GV1 : integer;

procedure main
    var a, b : integer;
    begin
        read from STDIN to a;
        GV1 := a + 2;
        call f(IN a, OUT b);
        write from b to STDOUT;
    end

procedure f(IN a : integer; OUT b : integer);
    var c : integer;
    begin
        c := GV1 - 4;                              — (a)
        if a > 0
            then
                begin
                    call g(IN c);                  — (b)
                    b := GV1 * 8;                  — (c)
                end
            else read from FILE1 to b;             — (d)
    end

procedure g(IN x : integer);
    var z : integer;
    begin
        z := x + 2;
        if x < 100
            then
                begin
                    call f(IN z, OUT y);
                    write from y to FILE2;
                end
            else read from FILE2 to GV1;
    end
```

Figure 1. An Example Program

The Symbolic Class Equations for main

$\underline{\text{f.implicit}} = \text{LOW}$ — (1)

$\underline{\text{f.a}} = \text{LOW} \oplus \underline{\text{STDIN}}$ — (2)

$\underline{\text{GV1}} = \text{LOW} \oplus \underline{\text{STDIN}}$ — (3)

$\underline{\text{STDOUT}} = \text{LOW} \oplus \underline{\text{f.b}}$ — (4)

The Symbolic Class Equations for f(IN a, OUT b)

$\underline{\text{b}} = \text{LOW} \oplus \underline{\text{a}} \oplus \underline{\text{GV1}} \oplus \underline{\text{FILE1}} \oplus \underline{\text{implicit}}$ — (5)

$\underline{\text{g.implicit}} = \text{LOW} \oplus \underline{\text{a}} \oplus \underline{\text{implicit}}$ — (6)

$\underline{\text{g.c}} = \text{LOW} \oplus \underline{\text{a}} \oplus \underline{\text{GV1}} \oplus \underline{\text{implicit}}$ — (7)

The Symbolic Class Equations for g(IN x)

$\underline{\text{f.implicit}} = \text{LOW} \oplus \underline{\text{x}} \oplus \underline{\text{implicit}}$ — (8)

$\underline{\text{f.z}} = \text{LOW} \oplus \underline{\text{x}} \oplus \underline{\text{implicit}}$ — (9)

$\underline{\text{FILE2}} = \text{LOW} \oplus \underline{\text{x}} \oplus \underline{\text{f.y}} \oplus \underline{\text{implicit}}$ — (10)

$\underline{\text{GV1}} = \text{LOW} \oplus \underline{\text{x}} \oplus \underline{\text{FILE2}} \oplus \underline{\text{implicit}}$ — (11)

Figure 2. The Symbolic Class Equations for the Example Program

```
procedure find_least_fixed_points
    begin
        for i := 1 to n do $Y_i$ := (LOW, { });
        repeat
            CHANGE := false;
            for i := 1 to n do
                begin
                    $new_i = f_i(Y_1, \ldots, Y_n)$;
                    if $new_i \neq Y_i$ then
                        begin
                            CHANGE := true;
                            $Y_i := new_i$;
                        end
                end
        until not CHANGE;
    end
```

Figure 3. Iterative Algorithm to Calculate the Least Fixed Points

An INFOSEC Platform

Joe Marino and Paul Lambert
Tactical Secure Communications Office
Motorola, Inc. Government Electronics Group

ABSTRACT

This paper describes the architecture and design approach taken by Motorola's Government Electronics Group in the development of a state-of-the-art information security (INFOSEC) products designed to bring computer and network security services to the new generation of automated information processing systems. The architecture is based upon a hardware and software platform that utilizes an open systems approach to the integration of cryptography into computer and communication systems. In this approach the cryptographic communication security (COMSEC) is supported by "open" specifications for both the hardware and logical software interfaces. The security protocols, key management techniques, and cryptography of the INFOSEC platform are based on NSA's Secure Data Network System (SDNS) standards. The first product, based on the INFOSEC Platform, is the Network Encryption System (NES) and is presently under evaluation through the Commercial COMSEC Endorsement Program. The NES products provide link and network layer security services for IEEE 802 local area networks.

INTRODUCTION

Strong assurances are required for the integration of Type I cryptographic security into computer and communication systems. INFOSEC systems must meet a variety of requirements that include those for COMSEC, COMPUSEC, TEMPEST, QUADRANT, and SFA. The difficulty of attaining the assurances and certifications inherent in these systems make it desirable to isolate the sensitive functionality. The Motorola INFOSEC platform has been developed for this environment. The services have been built into a single flexible architecture that provides uniform interfaces to access the security related functionality.

The design approach for the platform is based on an "open systems" philosophy. In this design approach, openly distributed standards are used for as much of the system as possible. For the platform, the hardware interfaces conform to VMEbus standards. The software interfaces for task-to-task and processor-to-processor are based on a widely distributed "common environment". The cryptography and key management services that the INFOSEC platform provides are in conformance with the recently developed Secure Data Network System (SDNS) specifications.

The goal of this open system approach has been to leverage existing technology in the development of secure computer and communication systems. The following are benefits of this design approach:

- Reduced government resources required for endorsement;
- Reduced development effort required to bring the security services to a broad population of computer and communications equipments;
- Reduced life cycle cost by adhering to standards initiatives, thereby lowering training, maintenance, and support costs;
- Increased interoperability by conforming to computer and communication standards.

INFOSEC PLATFORM HARDWARE ARCHITECTURE

The platform has been developed to support commercially available processor and communication products. The platform is based on the VMEbus specifications to allow any of a myriad of commercial board level products to be integrated into the system. A block diagram of the hardware is shown in Figure 1. A floppy disk is provided for configuration, software download, and audit purposes. The display is used for interactions with a local operator. The ignition key (IK) is used to enable and disable the system. In addition, this data key is used to provide the "seed key" for the system's cryptographic initialization.

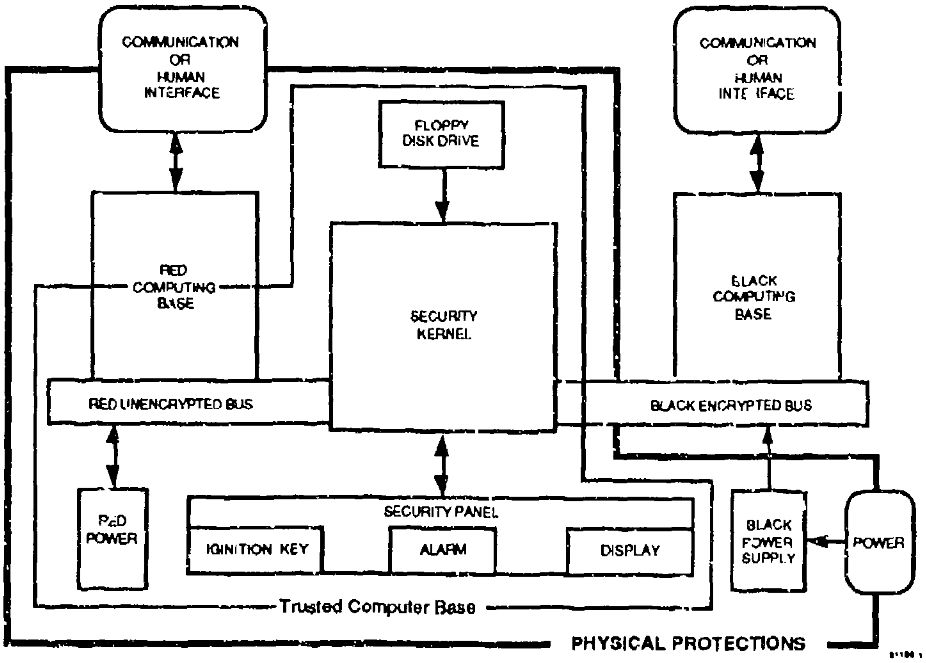


Figure 1. Motorola's INFOSEC Platform

The cryptographic security processes reside between the RED and BLACK computer busses in a centralized area of the device called the security kernel. This mechanism provides a strong separation of the RED and BLACK subsystems. The security kernel interfaces with the VMEbus using a commercially available intertask communication interface called the Common Environment. The Common Environment allows all software developed for the subsystems to be independent of the physical hardware. The security kernel contains many of the security relevant functions for the platform except for the RED Trusted Computer Base (TCB) and mechanical requirements needed for electromagnetic and physical protection. The security kernel architecture is based on custom VLSI, proven COMSEC design techniques, and provides the necessary security tools and assurances required to support the trusted elements on the RED unencrypted bus in a distributed computing environment.

The criteria normally defined for a reference monitor are that it must always be invoked, verified correct, and tamperproof. By this definition the platform's security kernel can be considered a hardware reference monitor which extends the system TCB by allowing it to directly access the cryptographic functions. The INFOSEC platform has been initially developed from a cryptographic and communication security (COMSEC) perspective. The kernel controls communications with peer systems and access to protected resources. The security kernel functionality also includes label checking associated with interface events and can be used to pass authenticated information to the RED side TCB.

UTILIZING SECURITY SERVICES WITHIN THE PLATFORM

The INFOSEC platform is designed for the protection of computer systems. The communication of peer entities is supported by security mechanisms modeled within the International Organization for Standardization (ISO) Open Systems Interconnect (OSI) reference model. Figure 2 shows the relationship of the platform services within the reference model and depicts alternatives for communication transfers. A secure computer or secure communication system will normally use several of the communication paths illustrated below in Figure 2.

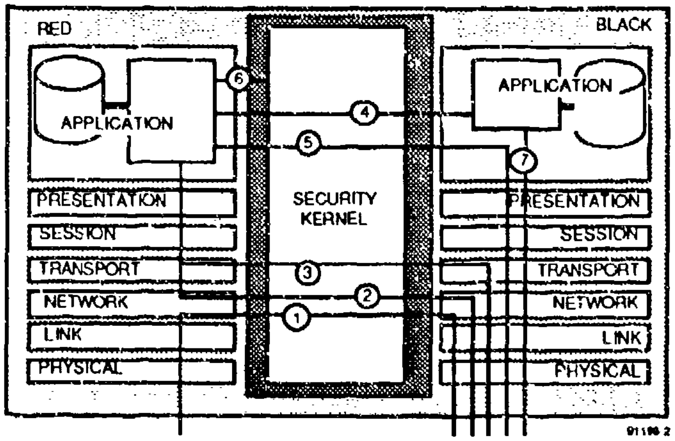


Figure 2. Security Services in the OSI Reference Model

The paths numbered 1, 2, and 3 illustrate examples of network and transport security mechanisms. Path 1 is an instance of the platform serving as a "router like" device using a network layer security protocol. Path 2 is also an example of applying network layer security to protect communications between peer applications running within the RED computer base. Transport layer security is normally colocated with the protected system and is shown by Path 3.

Security in the application layer of the OSI reference model is represented in Figure 2 by the paths numbered 4 and 5. The principle example for this mechanism is secure messaging based on extensions to CCITT X.400 electronic messaging. The secure messaging could be for a workstation built into the RED subsystem or the security could be provided to an enclave of users attached to the RED subsystem through a local media. File security for an operating system, or for file transfer protocols, are also possible.

Requests to the security kernel for key management or authentication services from the RED side protocols are conveyed via path 6. This path may prove useful to TCBs that can exploit cryptosealing, access control or authentication services, and also serves as the control interface for the kernel.

The BLACK computing base in path 7 does not utilize any local security services. This corresponds to a variety of real applications that require distribution of previously protected electronic mail messages, directory service capabilities, staging of encrypted files, or "BLACK" communication management. The "BLACK" communication management is required for interaction with systems that are not cryptographically protected.

ACCESSING PLATFORM SERVICES

The architectural design of the INFOSEC platform is based on a strong separation of the RED and BLACK computing subsystems. Communication between subsystems is mediated by the security kernel. Figure 3 illustrates that access to the security services are performed through four classes of interface commands. The interfaces support the basic functional operations of key management, system management, application control, and cryptographic (i.e., encryption, cryptoseal and etc.) functions.

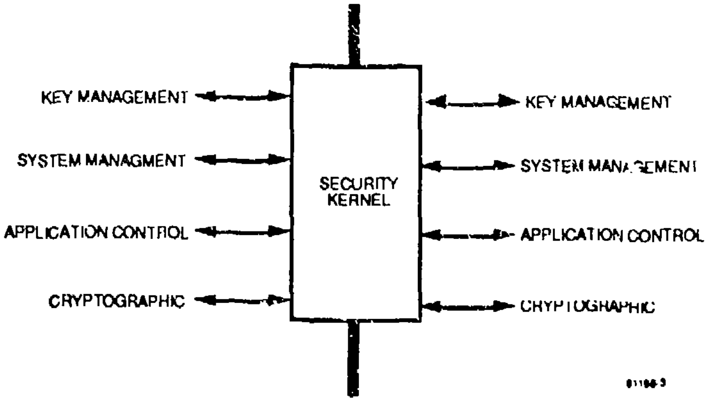


Figure 3. Kernel Command Interface

The cryptographic key management in the security kernel is based on SDNS specifications. The platforms key management interface allows cryptographic keys to be created, installed, transferred, archived, and destroyed. Cryptographic keys are isolated within the security kernel and controlled by the interface commands to provide a secure tamper-free environment for the TEK cache. The required key management protocois are contained within the security kernel.

The security kernels system management interface provides for operations that are loosely modeled after the OSI common management framework. These commands provide for system configuration, security audit, initiation of SDNS events (Rekey, get CKL), selftest, RED to BLACK flow control, and identity based access control functions.

The Application Control Service commands provide services for the establishment and termination of application associations which are identified by application titles. These associations are used by all kernel services that require external communications. The Application Control Services can be accessed through either the encrypted or unencrypted subsystems.

The cryptographic commands support the utilization of the kernels encryption and decryption hardware. These are the only commands that may be invoked on arbitrary user data and thus force the cryptographic protection on all data flowing through the kernel. The key management, systems management and application control of the security kernel do not allow user data to flow between the subsystems.

NETWORK ENCRYPTION SYSTEM (NES)

The Motorola NES is a Type I, COMSEC Controlled Item (CCI), data security device which provides security services based on standards developed by the Secure Data Network System . The initial IEEE 802.3 NES supports security services in both layers two and three of the OSI reference model. The link layer (layer two) security provides protection over IEEE 802 local area networks. This link layer protection does not extend over non802 media, but is useful in protecting systems based on proprietary network protocols.

The product also supports transparent operation over internets. This mode of protection offers true "end-to-end" security based on the SDNS - SP3 network layer security protocol. This protection is effective for both the DOD TCP/IP and the OSI Connectionless Network Layer Protocol (CLNP) internet environments.

The communication architecture of the NES allows it to be installed in a wide variety of network environments and topologies. An example of the NES communication environment is illustrated in Figure 4. This diagram shows the nature of the peer-to-peer protection provided by the NES. The BLACK networks can be either wide area point-to-point networks (WANS, i.e., X.25) or local area networks (LANS, i.e., 802.3). In this figure, the NES is shown as a front end and as an intermediate system. The BLACK internet refers to the portion of the network over which the NES provides protection. The RED networks are typically small groups of physically protected colocated computers.

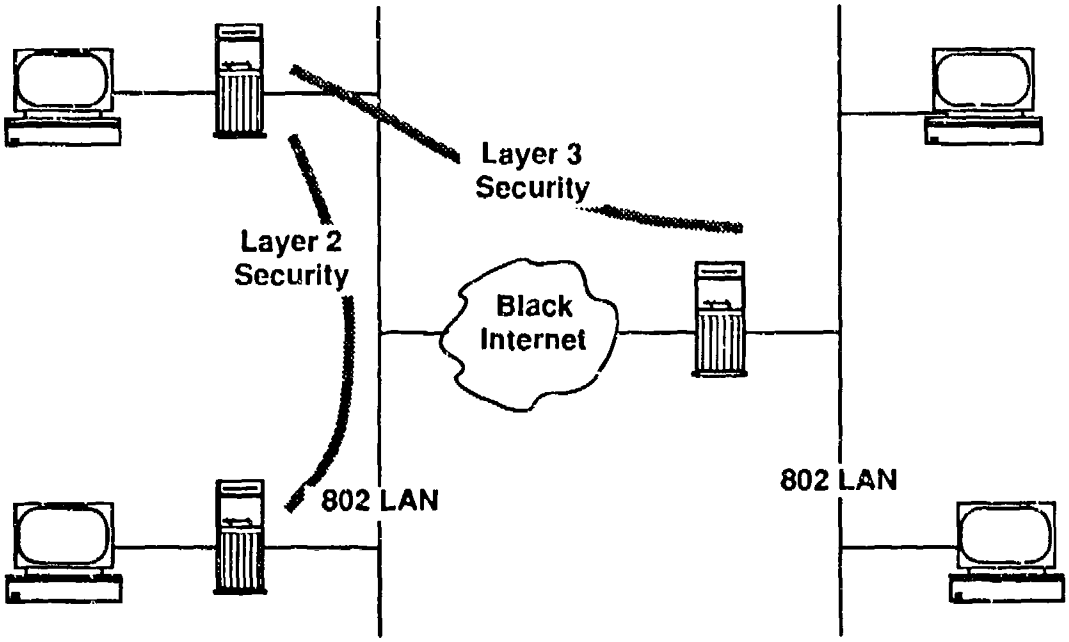


Figure 4. NES Network Environment

The connectivity of the initial NES is limited only by the requirement that the physical and link layer interfaces must conform to IEEE 802 standards. The NES provides link and network layer security services for 802 LANs. It is important to note that the network layer security mechanisms allow the protected BLACK internet to be composed of other network media besides IEEE 802. There is no maximum limit to the number of NES devices that can be installed in a communication system. The only limitations are the practical networking limitations imposed by a particular local medium. The security services that an NES provides are limited by the number of simultaneous cryptographic associations. Up to 250 simultaneous cryptographic associations are supported. In practice, this will typically mean that user traffic can be protected through an NES to 250 other NESs at any time. This limitation does not affect the number of instances of communication through a pair of NESs. A pair of NESs can support many pairs of communicating computer systems on a single cryptographic association.

The NES maintains a key cache for 256 cryptographic keys, with 250 for cryptographic associations. The remaining key cache (6 key entries) are used for key management functions, security functions, and selftests. One of the principal architectural considerations in the installation and maintenance of Motorola NES is the nature of the connectivity to the Key Management Center (KMC). SDNS security devices require infrequent communication with the KMC. The NES provides for connectivity through the public telephone network for all keying requirements except the initial seed key which is delivered by an approved KSD-64A key fill device.

PLATFORM KEYING SERVICES

The flow of keying information from the KMC to the NES is shown in Figure 5. Communication with the KMC is provided through the NES Portable Service Computer (PSC) and dial-up asynchronous links through the public telephone system. The transfer of the keying material is protected by the mechanisms defined by the Key Management Protocol (KMP). The NES PSC provides for a wide variety of maintenance services beyond keying services, including software maintenance, configuration, audit, access control, communication diagnostics, and security health diagnostics. The NES service computer allows for initial seed key conversion to operational key, allows for occasional rekey or operational material renewal, and allows for the replacement of operational material due to NES failure or maintenance. In summary, all TEKs are established using SDNS protocols and authentication mechanisms. All operational TEKs are stored and protected in the security kernel during use.

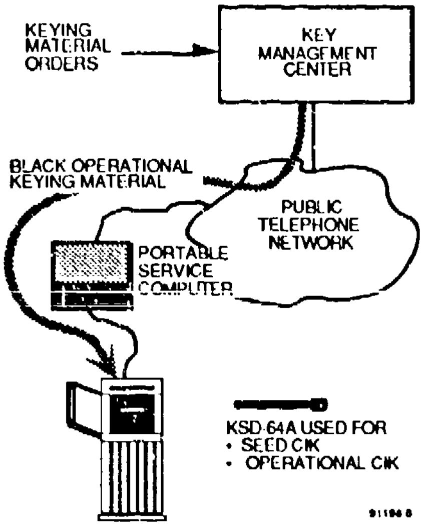


Figure 5. NES Keying

COMMUNICATION SERVERS

The NES is the first of a product line based on the INFOSEC platform. Future network layer security products are readily developed by the simple integration of new VME communication cards. Software integration of these systems is facilitated by the common environment logical interfaces. Communication products are envisioned that cover a broad range of networking applications. The SDNS specifications are rapidly gaining acceptance and so the SDNS defined SP4 transport layer security and electronic messaging protocol implementation are planned for future products. File security based on protocols like FTAM, or the network file sharing (NFS) protocol, will be valuable tools for securing networks of computers.

SECURE COMPUTERS

The INFOSEC platforms RED computer bus can readily support general purpose processing cards. By installing such a processor, the platform becomes a secure workstation. The cryptographic functionality of the security kernel is easily integrated using the same software interfaces used for the communication products. The installation of a hard disk on the BLACK side of the platform allows the system to support the encryption of files. Since the BLACK computer bus is outside the security boundary, a general purpose interface may be used and attached to any commercially available storage system. The installation of a communication capabilities in the BLACK subsystem allows the secure computer to utilize the SDNS security protocols. This approach for embedding computers inside a protected chassis should provide a valuable capability for the near-term development of secure systems. The cryptographic capabilities of a secure workstation based on the Motorola INFOSEC platform could be designed for a variety of missions including: key management applications, directory servers, file security, data base security, and electronic mail servers.

SUMMARY

The INFOSEC platform is unique in that it brings together communication and security standards, computers and INFOSEC design principles in a powerful secure communications platform. The initial product based on this architecture will provide a powerful tool for the protection of local area networks. Future applications will evolve the platform into secure computing environments with integrated cryptography.

REFERENCES

ISO 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model.

Saydjari, O.S., Beckman, J.M. and Leaman, J.R., "Locking Computers Securely," *Proceedings of the 10th National Computer Security Conference*, Baltimore, MD, October 1987.

SDN.301 Security Protocol 3, 1988-01-26.

SDN.401 Security Protocol 4, 1988-01-26.

Lambert, P., "Architectural Model of the SDNS Key Management Protocol," *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, October 1988.

A MULTILEVEL SECURE OBJECT-ORIENTED DATA MODEL

M.B.Thuraisingham
The MITRE Corporation, Bedford, MA

Abstract

A multilevel secure object-oriented data model, SO2, is described here. We first developed a multilevel type system and then defined a multilevel object-oriented database. It is this approach that could establish a theoretical framework for secure object-oriented systems. Also discussed here are the issues involved in (1) developing a security policy (2) handling polyinstantiation (3) using security constraints and (4) handling the inference problem for our model.

1. Introduction

Since its inception in 1970, the relational model [1] has enabled database designers to develop highly functional database management systems which have matured into successful products in the marketplace. Although it is the preferred choice of many database designers and researchers, the relational model has its limitations. The most notable one being that it views the world as a set of relations. In contrast, humans view the world not as a set of relations but principally as a set of objects [2].

Among the other models that have evolved over the years, object-oriented data models appear to have the features which address this problem. That is, an object model would enable conceptual entities to be represented as objects similar to our perception of the world. This power of representation has led to the development of new generation applications such as CAD/CAM, Image Processing, Artificial Intelligence and Process Control (see for example [3, 4, 5, 6, 7]). However the increasing popularity of Object-Oriented DBMS should not obscure the need to maintain security of operation. That is, it is important that such systems operate securely in order to overcome any malicious corruption of data as well as prohibit unauthorized access to and use of classified data especially with military applications.

It is only during the last two years that multilevel security has been incorporated into object-oriented data models. These include, among others, the following:

1) **SODA** - (Secure Object-Oriented DAtabase System) this is the first multilevel secure object-oriented database system for which a prototype has been developed. The data model for SODA is based on Smalltalk (Smalltalk was developed at XEROX Corporation [8]) with extensions to accommodate security concepts [9]. The prototype of this system is discussed in [10].
2) **HYPE*** - (Secure Object-Oriented Data Model for HYPErmedia Systems) this is a secure object-oriented data model developed for hypermedia systems [11]. This model incorporates security into the ORION [12] data model. A further discussion of this model is given in [13, 14].
3) **SOS*** - (A Simple Object-Oriented Secure Data Model) this is a simple secure object-oriented data model which is a restricted version of HYPE. This model uses security constraints to determine the levels at which the class constructs (or types) have to be created [15]. A design of the database system for query and update processing is given in [16].

In this paper we propose a multilevel secure object-oriented data model, SO2, which evolved from the data model O2 (O2** was developed by the Altäir group in France [17]). Unlike many of the other object-oriented data models that have been developed, O2 involved a type system defined in the framework of a set-and-tuple data model. That is, O2 accommodates tuple as well as set-based data structures allowing complex database objects to be represented. It is this approach that provides the foundations for establishing a theoretical framework for object-oriented systems [17]. With the help of this framework, we have incorporated multilevel security into O2.

The organization of this paper is as follows: In Section 2 we will describe concepts in multilevel secure database management systems (MLS/DBMS). In Section 3 we will give an informal overview of O2. In Section 4 we will describe SO2, a multilevel secure object-oriented data model. This model extends O2 by incorporating security properties. In Section 5 we will discuss the mandatory security issues in an object-oriented database system based on SO2. The paper is concluded in Section 6.

2. A Brief Account of Multilevel Secure Database Systems

In a multilevel secure database management system (MLS/DBMS) users cleared to different security levels access and share a database consisting of data at different sensitivity levels. The sensitivity levels (which we will also refer to as security levels) may be assigned to the data depending on content, context, aggregation and time. An effective security policy for MLS/DBMS should ensure that users only acquire the information to which they are authorized. The earliest of security policies, the Bell and LaPadula security model [18], is not sufficient to ensure multilevel security in a DBMS as users can pose multiple queries and infer unauthorized information from the legitimate responses that they receive. Despite its shortcomings, extensions to the Bell and LaPadula security model have since been proposed for some MLS/DBMSs [see for example 19].

** HYPE and SOS are names given only in this paper for convenience to differentiate between the above models*

*** In [17], O2 is denoted by O_2*

The relational data model has dominated much of the work on MLS/DBMSs [for example 20, 21, 22, 23, 24]. As a result of such work, multilevel secure relational database systems have been developed not only as prototypes but also as products [25]. In recent times security issues have also been investigated in other systems such as entity relationship systems [26], object-oriented systems [9] and knowledge-based systems [27] among others. A detaiied description of the recent development in database security is given in [28].

3. Overview of O2

Following [17], in O2 an object could be either a basic object, a tuple-object or a set object. An object consists of an identifier and a value. Examples of objects are given below:

(ob1, "Smith") ; basic object
(ob2, 15) ; basic object
(ob3, <name: "Smith", age: 32>) ; tuple object
(ob4, <name: "John", age: 28, salary: 20K>) ; tuple object
(ob5, {ob3,ob4}) ; set object

In O2, a type is represented by a name, a structure and a set of methods. Methods are applied to the objects of a type. A structure is either a basic structure, a tuple structure or a set structure. Basic structures include String, Integer, Real and Boolean. Examples of tuple and set structures are given below:

Person = <name: String, age: Integer, sex: String> ; tuple structure
Persons = {Person} ; set structure
Employee = <name: String, age: Integer, sex: String, salary: Integer> ; tuple structure
Employees = {Employee} ; set structure

Structures have interpretations. The interpretation of, say, the structure Person includes any entity which has a name, age and sex. Similarly the interpretation of Employee includes any entity which has a name, age, sex and salary. The interpretation of Employee is included in the interpretation of person. Therefore Employee is a substructure of Person. That is, every employee is a person. This is the IS-A relationship where an employee inherits all of the properties of a person. Inheritance also applies to methods. As employee is a person, any method which can be applied to a person can also be applied to an employee.

4. SO2 - A Secure Object-Oriented Data Model

SO2 has evolved from O2 by incorporating security levels for all entities and enforcing security properties that must be satisfied. The issues are discussed in this section. Thus in Section 4.1 are described multilevel universe of objects. Multilevel universe of types will be discussed in Section 4.2. Note that there are two components to a type: a structure and a set of methods. The structure defines the data structures of a type and the methods define the operations on a type. The notion of a multilevel type system will be introduced in Section 4.3. Then in Section 4.4 we will define a multilevel object-oriented database.

4.1 Multilevel Universe of Objects

The universe consists of a set of objects An object consists of an identifier and a value and is defined by :

o = (i,v) where o is the object defined, i is the identifier of the object and v is the value of the object.

The security properties that must be satisfied are given below:

P1: If o is an object, then there is a level L such that Level(o) = L
P2: If object o= (i,v) where i is the identifier and v is the value, then Level(o) >= l.u.b.(Level(i), Level(v))
where Level(x) is the security level of x.

In general we denote an object by a triple (i,v,L) where i and v are as before and L is the security level of o. In order to complete the definition of an object, we need to define what we mean by an identifier and a value of the object. These definitions are given below:

Domains, Attributes and Identifiers

D1, D2, ...Dn (n >= 1) are a set of finite domains. D is the union of all domains. The following security properties should be satisfied:

P3: If Di is a domain, then there is a level L such that Level(Di) = L
P4: If x belongs to a domain Di, then Level(x) >= Level(Di)

A is a countably infinite set of attributes. The following security property is associated with an attribute:
P5: If a belongs to A, then there is a level L such that Level(a) = L

Let ID be a countably infinite set of symbols called identifiers. The following security property is associated with identifiers.

P6: If id belongs to ID, then there is a level L such that Level(id) = L

Values

There are three types of values: basic values, tuple values and set values. Let V be the set of all values. The following security property is associated with V:

P7: If v belongs to V, then there is a security level L such that Level(v) = L

Each type of value will be described below.

Basic values:

(i) Special symbol nil is a basic value.
(ii) Each element x of D is a basic value.

The security properties of basic values are:

P8: Level(nil) = system-low (In military environments the system-low level is usually the Unclassified level)
P9: If value v is the element x of D, then Level(v) = Level(x)

Set values:

Every finite subset of ID is a set value. The following security property holds:

P10: If V is the set value {id1,id2,......idn}, then Level(V) >= l.u.b.(Level(id1), Level(id2) ....Level(idn))

Tuple values·

A tuple value is a partial function f from A into ID. It is denoted by <a1:i1, a2:i2, ...ap:ip> where f(ai) = ip for all i.
The following security property holds:

P11: Level(f) >= l.u.b(Level(a1), Level(i1), Level(a2), ...Level(ip))

Objects Revisited

Depending on the kind of value that is used to define an object, an object can be either a basic object, a set object or a tuple object. The set of all objects O is ID x V where V is the set of all values.

Note that in our definition of an object, we have assumed that the security level of the object could dominate the security level of its identifier. This means that two different objects at different security levels can have the same identifier. This is a form of polyinstantiation in object-oriented systems. We will address this problem in a later section.

We use the following notations. If o = (i,v), then i = ident(o) and v = value(o). The function from O (the set of all objects) to 2^{ID} (the set of all subsets of ID) will be denoted by ref. That is, ref(o) is the set of identifiers referenced by the object o.

We define two objects to be identical only if the following condition is satisfied:

two objects o1 = (i1,v1) and o2 = (i2,v2) are identical, if i1 = i2 , v1 = v2 and Level(o1) = Level(o2).

Graphical Representation

Objects can be represented graphically using an object graph. Let TH be a multilevel set of objects. The representation of TH at a security level L is graph(TH, L) which consists of nodes and links at the security level L. The complete graph of TH is union of all graphs at the various security levels. Graph(TH,L) can be obtained as follows:

(i) If o is a basic object of TH whose security level is dominated by L, then the graph contains a vertex represented by (@) and is labelled with the identifier of o and the security level of o. The value of o is also attached to this vertex.

(ii) If o is a tuple structured object (i, <a1:i1, a2:i2.........ap:ip>) of TH whose security level is dominated by L, then the graph of o contains a vertex, say, v represented by a dot (•) and labelled with i and the security level of o. Furthermore, for each ak, there is an edge from this vertex. The edge is labelled with ak and the security level of o. The tail of the edge is the vertex labelled by ik and the security level of the object whose identifier is ik.

(iii) If o is a set object (i, {i1,i2,......ip}) of TH whose security level is dominated by L, the graph of o contains a vertex represented by (*) and labelled by i and the security level of o. For each ik, there is an edge from this vertex to a vertex labelled ik. This edge is labelled with the security level of o.

Example of Objects

Let TH be a set of objects with the following members:

o1 = (i1, <spouse:i2, name:i4, children:i3>, Secret)
o2 = (i2, <spouse:i1, name:i5, children:i3>, Secret)
o3 = (i3, {i6,i7}, Unclassified)
o4 = (i4, "Fred", Unclassified)
o5 = (i5, "Mary", Unclassified)
o6 = (i6, "John", Unclassified)
o7 = (i7, "Paul", Unclassified)

The graphical representation of TH is shown in Figure 1. In this figure, the Secret nodes are circled and the Secret links are represented by darkened lines.

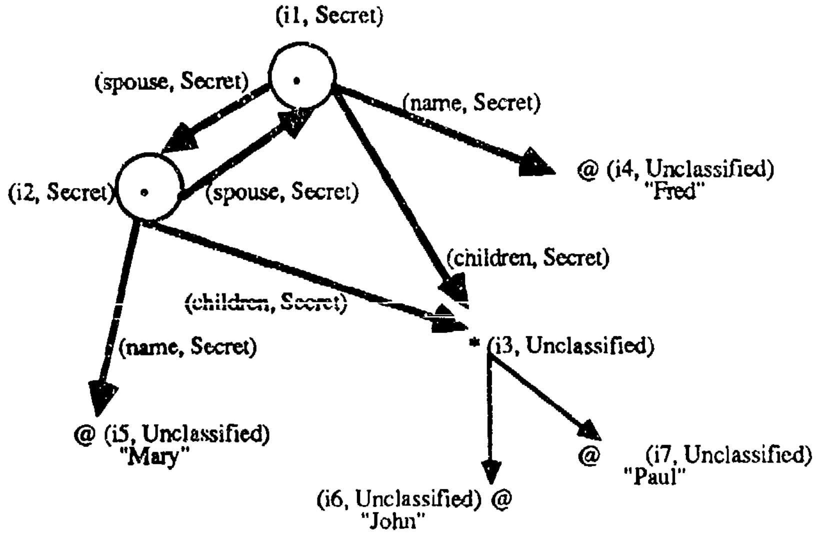


Figure 1 - Graphical Representation of Objects

Consistent Objects

We enforce consistency within a security level. The objects could be inconsistent across security levels as it is possible for two objects at different security levels to have the same identifier. We also ensure that all of the objects that are referenced at a security level are visible at that level.. In other words, there are no dangling references in the graphical representation of objects.

We define a set of objects TH to be consistent at a security level L if the following three conditions are satisfied:

(i) The set of all objects in TH at levels dominated by L is finite.

(ii) The ident function is injective within L. That is, no two objects which are classified at the security level L have the same identifier.

(iii) For all o in TH at a level dominated by L, ref(o) $\subseteq$ ident(TH, L) where ident(TH,L) is the set of all identifiers of objects in TH whose security levels are dominated by L.

4.2 Multilevel Universe of Types

Types enable data and operations to be encapsulated in the same structure. Like objects, identifiers, domains, values and attributes, types are also entities of classification. That is, each type is assigned a security level. A type consists of a type structure and a set of methods. The type structure specifies the structure associated with the type. The methods specify the operations that are defined on the type. In this section we will define the concepts of type names, types, type structures, schema, methods and interpretations of type structures and methods.

Type Names

There are two kinds of type names: Bnames (basic names) and Cnames (constructed names). The union of Bnames and Cnames are denoted by Tnames. Each type name is assigned a security level. Cnames is the set of names for constructed types which is countably infinite and disjoint with Bnames. Bnames is the set of names for basic types and contains the following:

(i) The special symbols Any and Nil.
(ii) A symbol di for each domain Di.
(iii) A symbol 'x for each value x of D.

The following security properties are associated with Type Names.

P12: Level(Any) = system-low and Level(Nil) = system-low
P13: Level(di) = Level(Di)
P14: Level('x) = Level(x)
P15: If c belongs to Cnames, then there is a level L such that Level(c) = L

Types

A type consists of a type structure and a set of methods. The security level of the type is that of the type structure associated with it. The set of methods associated with a type could have a different security level. We will discuss the notion of a method later. For now we assume that MT is the finite set of all methods and each type will have a subset of MT associated with it.

There are two kinds of types: basic and constructed. Each of these kinds will be described below. A basic type (Btype) is a pair (n,m) where n is an element of Bnames and m a subset of MT. A constructed type is one of the following:

(i) (s=t,m) where s is an element of Cnames, t is an element of Tnames and m a subset of MT (s=t is the type structure associated with this type. The structure of a type s is denoted by struct(s)).
(ii) (s=t, m) where s is an element of Cnames, t is a partial function form A (the set of attributes) into ID and is represented by <a1:s1, a2:s2,.........ap:sp> where t(ak) = sk for all k and m is a subset of MT. The type defined this way is a tuple structured type (s=t is the type structure).
(iii) (s = {s'}, m) where s is an element of Cnames, s' is an element of Tnames and m is a subset of MT. The type defined this way is called a set structured type (s = {s'} is the type structure).

A type is either a basic type or a constructed type. The set of all types is denoted by T. The following is the security property associated with a type.

P16: If s is a type, then Level(s) = Level(struct(s))

Type Structures

There are two type structures: basic and constructed. If t = (n,m) is a basic type, then n is the basic type structure associated with this type. The security property associated with this type structure is:

P17: Level(n) is the level assigned to the Bname n

Let t = (s=x, m) be a constructed type. Then s=x is the constructed type structure associated with t. The following security properties hold:

P18: If t is neither a tuple structured type nor a set structured type, then Level(strcut(t)) >= Level(x)
P19: If t is a tuple structured type, and x is of the form <a1:s2,a2:s2,.......ap:sp>, then
Level(struct(t)) >= l.u.b(Level(a1), Level(s1),. . . . . . . .Level(ap),Level(sp))
P20: If t is a set structured type and x is of the form {s'}. then Level(struct(t)) >= Level(s')
In general we will denote a type structure s by s = (structure definition, security level).

Schemas

We first need the following notations: If t is a type, then name(t) is the name of the type. If st is a the type structure associated with a type t, then the name of the type structure st is denoted by name(st). This name is the same as name(t). If st is the type structure associated with t, refer(st) is the set of types referenced by st either directly or indirectly.

A set DELT of constructed type structures is a schema at level L if and only if the following conditions are satisfied:

(i) The set of type structures in DELT whose security levels are dominated by L is finite.
(ii) The name function is injective within L; that is there are no two type structures with the same identifier assigned at the security level L.
(iii) For all structures st which belong to DELT where the security level of st is dominated by L,
refer(st) $\cap$ Cnames $\subseteq$ name(DELT,L) , where name(DELT,L) is the set of names of type structures in DELT whose security levels are dominated by the security level L.

The set DELT is a multilevel schema, if it is a schema at every security level. For example, let DELT consists of the following type structures:

age = (Integer, Secret)
person = (<name:String, age: age>, TopSecret)

DELT is a schema at every security level. However, if we had classified age at the TopSecret level and person at the Secret level, then DELT will not satisfy the properties of being a multilevel schema.

Interpretation

Let a type structure s have security level L1. Intuitively, an interpretation of s at a security level L (>= L1) is any set of objects where each object in this set is classified at any security level between L1 and L (both L1 and L inclusive) such that the structure of the object is included in the structure of s. By a structure s1 being included in another structure s2 we mean either s1 and s2 are identical or s1 has all the components of s2 plus some additional components. That is, any object of structure s1 is also of structure s2 (the converse is not necessarily true). A formal definition of interpretation is given below.

Let DELT be a multilevel schema and TH be a consistent subset of the multilevel universe of objects. An interpretation I of DELT at security level L in TH is a function from Tnames to $2^{\mathrm{ident}(TH)}$, satisfying the following properties:

Basic Type Names:

(i) $I(Nil, L) \subseteq \{ i \in (ident(TH),L) \mid (i,NIL) \in TH\}$ where (ident(TH),L) is the set of all identifiers of objects in TH which are dominated by L.
(ii) If Level(di) <= L, then $I(di,L) \subseteq \{ id \in (ident(TH),L) \mid TH(id) \in (Di,L)\} \cup I(NIL,L)$ where (Di,L) is the set of all values in a domain Di whose security levels are dominated by L, and TH(id) is the value of the object whose identifier is id.
(iii) If Level(x) <= L, then $I(di,L) \subseteq \{ id \in (ident(TH),L) \mid TH(id) = x$ and $Level(x) <= L\} \cup I(NIL,L)$

Constructed Type names:

(iv) If s = <a1:s1, a2:s2, ...ap:sp> is in DELT and its security level is dominated by L, then
$I(s,L) \subseteq \{ id \in (ident(TH),L) \mid TH(id)$ is a tuple structure value defined on (at least) a1,a2,....ap such that
$TH(id)(ak) \in I(sk,L)$ for all $k\} \cup I(Nil,L)$
(v) If s = {s'} is in DELT and its security level is dominated by L, then $I(s,L) \subseteq \{ id \in (ident(TH),L) \mid TH(id) \subseteq I(s',L)\} \cup I(Nil,L)$
(vi) If s = t is in DELT and its security level is dominated by L, then $I(s,L) \subseteq I(t,L)$.

Undefined Type Names:

(vii) If s is neither a name of a basic type nor a name of the schema DELT, then $I(s,L) \subseteq I(Nil,L)$.

Model of a Schema

A model of a schema at a security level L is defined by defining models at security level L of all type structures belonging to the schema Intuitively, a model of a type structure s (whose security level is L1) at security level L is the largest interpretation of s at L with respect to a set of objects. Formal definition of model of a schema is given below.

One can define a partial order on interpretations as follows:

An interpretation I is smaller than an interpretation I' if and only if, for all s which belongs to Tnames and security level L, $I(s,L) \subseteq I'(s,L)$.

If DELT is a multilevel schema and TH is a consistent set of objects (where consistency is only within a security level), the model M of DELT in TH at security level L is the greatest interpretation of DELT in TH at level L.

If s is a constructed type structure, then M(s,L) is the model of s at security level L. Note that this model is defined only if the security level of s is dominated by L. Furthermore, this model consists of all objects in TH between the security level of s and the security level L which have the structure of s. Informally, the model of a type structure will consist of all instances of that type.

The following security properties will hold:

P21 : If s is a type structure whose security level is L, and o belongs to M(s,L), then L* <= Level(o) <= L*
This property shows that the security level of an instance of a type dominates the security level of the type.

Partial Order Among Type Structures

Let s and s' be two type structures. A partial order $\leq_{st}$ among the type structure can be defined as follows:

$s \leq_{st} s'$ if for every security level L, $M(s,L) \subseteq M(s',L)$.

The following security property holds:

P22: If s $\le_{st}$ s' for any two type structures, then Level(s) >= Level(s')

An Example on Models of Type Structures

Let TH be the following set of objects:

{(i0, NIL, Unclassified), (i1, {i2,i3}, Secret), (i2, 1,Unclassified),
(i3, 4, Unclassified), (i4, <a:i2>, Unclassified), (i5, <a:i2, b:i3>, Secret)}

Let S be the following set of type structures:

{s1 = (<a:Integer>, Unclassified), s2 = (<a:Integer, b:Integer>, Secret),
s3 = ({Integer}, Unclassified)}

Then the following are the models of the type structures in S with respect to TH at various security levels:

M(s1,Unclassified) = {i0, i4}
M(s1,Secret) = {i0, i4, i5}
M(s2,Unclassified) = {i0}
M(s2,Secret) = {i0, i5}
M(Integer, Unclassified) = {i0, i2, i3}
M(s3,Unclassified) = {i0}
M(s3, Secret) = {i0, i1}

It can be seen that s2 $\le_{st}$ s1.

Methods

A method m is defined to be the pair (n, sig) where n is the name of the method and sig is a signature. The following security property holds:

P23: If method m = (n,sig), then Level(m) >= l.u.b. (Level(n), Level(sig)).

Next we need to define what is meant by a signature. Let DELT be a schema, then a signature over DELT is an expression of the form: s1 x s2 x ......sn -> s where s1,s2,....sn,s are all type structures in DELT. The following security property holds:

P24: If a signature sig is of the form s1 x s2 x ......sn -> s, then Level(sig) >= l.u.b.(Level(s1), Level(s2),Level(s3),......Level(s))

If m = (n,sig), then sig is a method defined on the first type in its definition. In the above example, it is s1. The following security property holds:

P25: If sig is a signature defined on type structure s, then Level(sig) >= Level(s)

Interpretation of Signatures

The model of a signature sig is defined as follows:

Let DELT be a multilevel schema and sig be a signature over DELT. Define sig to be s1 x s2 x .....sn -> s. Let TH be a consistent set of objects (within a security level). If the security level of sig is dominated by L, then the model of sig in TH at level L, denoted by M(sig,L) is the set of all partial functions from M(s1,L) x M(s2,L) x ....M(sn,L) -> M(s,L).

A partial order can be defined among signatures as follows: Let DELT be a multilevel schema and sig1 and sig2 be two signatures. Then sig1 is smaller than sig2 denoted sig1 $\le_{m}$ sig2, if for all security levels L, M(sig1,L) $\subseteq$ M(sig2,L).

An Example on Interpretation of Signatures

Let DELT be a schema consisting of the following type structures:

Person = (<name:String, age:Integer, sex: String>, Unclassified)
Persons = ({Person}, Unclassified)
Employee = (<name: String, age:Integer, sex:String, salary:Integer>, Secret)
Employees = ({Employee}, Secret)

Let SIG consist of the following two signatures:
sig1: Persons x Person -> Boolean
sig2: Employees x Employee -> Boolean

Let TH be a set consisting of the following objects:

(i0, NIL, Unclassified), (i1, <name: i6, age: i7, sex: i8>, Unclassified),
(i2, <name: i9, age: i10, sex: i11>, Unclassified),
(i3, <name: i12, age: i13, sex: i8, salary: i15>, Secret),
(i4, <name: i16, age: i17, sex: i11, salary: i18>, Secret),
(i5, {i1,i2}, Unclassified), (i19, {i3,i4}, Secret),
(i6, John, Unclassified), (i7, 28, Unclassified), (i8, Male, Unclassified),
(i9, Mary, Unclassified), (i10, 25, Unclassified), (i11, Female, Unclassified),
(i12, James, Unclassified), (i13, 40, Unclassified), (i15, 20K, Unclassified),
(i16, Jane, Unclassified), (i17, 35, Unclassified), (i18, 30K, Unclassified),
(i20, True, Unclassified), (i21, False, Unclassified)

Models of the signatures sig1, sig2 at various security levels are as follows:

M(sig1, Unclassified)
= set of all partial functions from : M(Persons, Unclassified) x M(Person, Unclassified) into M(Boolean, Unclassified)
= set of all partial functions from {i0,i5} x {i0, i1,i2} into {i0, i20, i21}

M(sig1, Secret)
= set of all partial functions from M(Persons, Secret) x M(Person, Secret) into M(Boolean, Secret)
= set of ail partial functions from {i0,i5,i19} x {i0,i1,i2,i3,i4} into {i0, i20, i21}

M(sig2, Unclassified)
= set of all partial functions from M(Employees, Unclassified) x M(Employee, Unclassified) into M(Boolean, Unclassified)
= set of all partial functions from {i0} x {i0} into {i0, i20, i21}

M(sig2, Secret)
= set of all partial functions from M(Employees, Secret) x M(Employee, Secret) into M(Boolean, Secret)
= set of all partial functions from {i0, i19} x {i0, i3, i4} into {i0, i20, i21}

It can be seen that sig2 $\leq_m$ sig1.

4.3 Multilevel Type System

A subset PI of the multilevel universe of types if and only if:

(i) the set of structures associated with PI is a multilevel schema
(ii) for all types t in PI and for all methods m in Methods(t) (which is the set of all methods associated with t), m is defined on struct(t).

We can now define the notion of a subtype as follows: If t and t' are types in PI, then t is a subtype of t' (denoted t ≤ t') if and only if:

(i) struct(t) $\leq_{st}$ struct(t',
(ii) for every method m' of t', there is a method m of t such that name(m) = name(m') and sig(m) $\leq_m$ sig(m'). (sig(m) is the signature portio of a method m.)

The following security properties hold among types and subtypes:

P26: If t is a subtype of t', then Level(t) <= Level(t)
P27: If t is a subtype of t', m $\in$ Methods(t), m' $\in$ Methods(t') and name(m)=name(m'), then
Level(m) = l.u.b(Level(m'), Level(t))

Property 26 states that the security level of the subtype should dominate the security level of the supertype. Property 27 deals with inheritance of methods. That is, the security level of an inherited method is the least upper bound of the security level of the original method and the security level of the subtype on which the inherited method is defined. It does not make sense to classify the inherited method at a higher level than the original method. In an object-oriented model, the inherited method is the same as the original method. However, in a secure object model the inherited method may not have the same security level as the original method. This is because the security level of the subtype on which the inherited method is defined could be higher than that of the original method and security property P25 ensures that the security level of a method should dominate the security level of the type on which it is defined.

When multiple inheritance is permitted, some additional security properties have to be introduced in order to resolve conflicts. Multiple inheritance is still a research issue in our work on multilevel object-oriented databases. Although the model defined here does not prohibit multiple inheritance, initially we assume that the multilevel database does not permit multiple inheritance. This multilevel database is defined in the next section.

4.4 Multilevel Database

A multilevel database is a tuple (PI, TH, <db, {ext-L}, {impl-L}> where

(i) PI is a multilevel type system with the associated multilevel schema DELT.
(ii) TH is a consistent set of objects (consistency within a security level).
(iii) <db is a strict partial order among the types in PI.
(iv) {ext-L}, for each security level L, an interpretation ext-L of DELT in TH at L.
(v) {impl-L}, for each security level L, a function impl-L, which assigns a function to every method m of a type t at level L.

The additional restrictions imposed in O2 in the definition of a database can be extended for a multilevel database as follows:

(vi) t $<_{db}$ t' implies t ≤ t'.
(vii) If t $<_{db}$ t' and t $<_{db}$ t'', then t' and t'' are comparable.
(viii) TH = $\cup_{\text{all L}}$ $\cup_{\text{t in PI}}$ ext-L(t)
(xi) ext-L(t) ∩ ext-L(t') = Φ for all L, if t and t' are not comparable. That is, an instance cannot belong to two types if one is not a subtype of the other. Note that it is assumed here that multiple inheritance is not permitted.
(x) If t is a type of PI and m a method of t having signature t x s2 x s3 ....sn -> s, then impl-L(m) is a function defined at least from ext(t,L) x ext(s2,L) x ....ext(sn,L) -> ext(s,L).

Note that the rule (vi) stated above means that the ordering <db implies the ordering ≤. The converse is not necessarily true. This is because the ordering <db is user defined. The user can define this ordering only if it is permitted in the model which is the ordering ≤. However, if the converse is also true, then some meaningless orderings will be defined. As stated in [17] an example is:

Age = (Integer, {+,-})
Weight = (Integer, {+,-})
According to the model Age ≤ Weight and Weight ≤ Age. But this ordering is meaningless. Therefore the user will not permit the following orderings:
Age $<_{db}$ Weight and Weight $<_{db}$ Age

5. Mandatory Security Issues in an SO2-Based Object-Oriented Database System

In this section we will describe the mandatory security issues in an object-oriented database system which is based on SO2. In Section 5.1 we will describe our mandatory security policy. Polyinstantiation issues will be describe in Section 5.2. Finally in Section 5.3 we will describe how security constraints which assign security levels to the data may be handled.

5.1 Security Policy

The security policy for an object-oriented database system based on SO2 consists of the following properties:

(i) Subjects and entities (we use the term entity instead of an object as it is usually stated in security policies in order to not confuse between the object in security policies and the object in an object-oriented system) are assigned security levels.
(ii) A subject has read access to any entity if the subject's security level dominates the security level of the entity.
(iii) A subject has write access to an entity, if the subject's security level equals the security level of the entity.
(iv) A subject can execute a method if the subject's security level dominates both the security level of the method and the type on which the method is defined.
(v) A method executes at the security level of the subject who initiated the execution.
(vi)During the execution of a method m1, if another method m2, has to be executed, then m2 can execute only if the execution level of m1 dominates both the security level of m2 and the security level of the type on which m2 is defined.
(vii) If a new object has to be created as a result of executing a method, the object is created at the security level of the subject who initiated the execution of the method.

Property (ii) is the simple property specified in the Bell and LaPadula security policy. Property (iii) is different from the *-property because writeup is not permitted (this is because it does not seem natural for a subject to write some data and not be able to read it later). The remaining properties are enforced due to method execution.

5.2 Polyinstantiation

Polyinstantiation generally occurs when two subjects at different security levels give different values or structures to represent in the database a single entity in the real world. However, in the case of object-oriented models, there is another form of polyinstantiation where two subjects at different security levels could also use the same identifier to represent two different entities in the real world. The entities that could be polyinstantiated are the object:. types and the methods. Here, we only discuss polyinstantiation among the objects. It should be noted that polyinstantiation is still a major research issue for us. Therefore much remains to be done before satisfactory solutions to handle polyinstantiation can be given.

Polyinstantiation occurs when:

(i)) an Unclassified subject has created an object, say o1, and a Secret subject creates a second object, say, o2 to represent the same entity and the Secret subject gives a different value or structure to the object created.

(ii) a Secret subject has created an object o1. The Unclassified subject is unaware of the existence of o1 and it creates another object to represent the same entity in the real world.The structure or value of the object created by the Unclassified subject may be different from those of o1.

(iii) an Unclas ified subject has created an object, say, o1 with identifier i1 and a Secret subject uses the same identifier i1 to represent a different entity in the real world.

(iv) a Secret subject has created an object, say, o1 with an identifiei i1 and an Unclassified subject uses i1 (which we assume is an Unclassified identifier) to represent a dif' rent entity in the real world.

A possible solution to handle the various types of polyinstantiations could be the following:

(1) A Secret subject requests to use the same identifier that is already used for an Unclassified object only when it wants to polyinstantiate the Unclassified object. Otherwise a different identifier is used.

(2) When a Secret subject creates an object (which is not a polyinstantiated object) then the Secret subject should use a Secret identifier for that object.

(3) If an Unclassified subject wants to create an object, say, o1 to represent the same entity which is already represented by a Secret object say o2, then the Unclassified subject will use an Unclassified identifier for o1. By 2), this will be different from the Secret identifier used by o2. However, with this approach there is no way to determine that o2 is a polyinstantiated version of o1 (unless we introduce the notion of primary key of an object which is not part of an object model).

We can justify (3) by taking Reiter's Closed World Assumption (CWA) [29] into consideration. CWA states that information is represented in the database if and only if it is true in the real world. Therefore for an entity to be represented by some Secret object and not by an Unclassified object means that the entity which exists in the Secret world does not exist in the Unclassified world. For the entity to be brought into the Unclassified world it has to be downgraded (by some trusted subject). Then the Secret object which represents the entity must be deleted as the entity is now in the Unclassified world. An Unclassified object is created to represent this entity. However, this same entity can have different values or structures in the Secret world. Then a Secret object can be created later to represent the same entity with the same identifier as that of the Unclassified object. With the solution that we have proposed we do not have to handle the case where two subjects at different security levels request the same identifier for two different objects which represent two different entities.

5.3 Security Constraints

Security constraints have been used in the past to assign security levels to the data [20]. The entities in our model are assigned security levels by using security constraints. However, the security levels assigned to the entities must satisfy the security properties. For example, if there is a type structure EMP = <name:String, age:age>, where the Integer type is Unclassified and age type is Secret, then one cannot have a security constraint which classifies EMP at confidential. This is because, the security property P will ensure that EMP is classified at least at the Secret level. However, the security constraint could classify EMP at the TopSecret level.

In the model SOS [15], the security constraints are used to create the various types. This technique can also be used to create types in a system based on the model SO2. For example, conside the type EMP = <name:String, salary: Integer, SS#: Integer>. This type will have as its instances, all employees. Suppose an Unclassified user should not see the names of the employees. This is a security constraint which is used by the schema manager to create certain types. One possibility will be to create two types EMP1 and EMP2 as follows:
EMP1 = (<salary"Integer, SS#:Integer>, Unclassified)
EMP2 = (<name:String, salary: Integer, SS#: Integer>, Confidential).
Note that EMP 1 is unclassified and EMP2 is confidential. Furthermore, EMP2 can be made a subtype of EMP1.

Another example is the following constraint: all salaries more than 50K are Secret while salaries less than or equal to 50K are Unclassified. In this case, three types EMP1, EMP2 and EMP3 are created.
EMP1 = (<name:String, SS#:Integer>, Unclassified),
EMP2 = (<name:String, salary:51K..200K, SS#:Integer>, Secret)
EMP3 = (<name:String, salary:0..50K, SS#:Integer>, Unclassified).
EMP1 and EMP3 are Unclassified while EMP2 is Secret. Furthermore, EMP2 and EMP3 can be made subtypes of EMP1.

5.5 Inference Problem

Security violations via inference occurs when users pose multiple queries and acquire unauthorized information [30, 31]. A solution to handling the inference problem in relational systems is to augment a relational DBMS with a logic-based inference engine and a knowledge base. The inference engine will detect security violations via inference when processing queries [32, 33, 34, 35]. A similar inference controller can be built for object-oriented systems also [36]. Two approaches to implementing such an inference controller are as follows. In the first approach, the database as well as the security constraints are expressed in a logic programming language with support for objects. An example of such a language is object-prolog [37]. In the second approach, an object-oriented database system is augmented with an inference engine and a rule base. The inference engine is based on an extension to first order logic. The queries are modified first by the inference engine before the object-oriented DBMS processes them. The techniques proposed in this second approach can be used to augment an SO2-based object-oriented database system with a logic-based inference engine which will detect security violations. Another direction in the investigation of the inference problem is to consider it as a decision problem for a deductive system and analyze its complexity [38].

6. Conclusion

We have developed a multilevel secure object-oriented data model, SO2, which has evolved from an object model O2 and we have also described its essential features with examples. Like O2, SO2 involved a type system that accommodates both tuple and set-based data structures. This has enabled us to develop SO2 based on a multilevel type system. It is this approach that provides the foundations for establishing a theoretical framework for secure object-oriented systems.

We have also discussed mandatory security in an object-oriented system based on SO2. We first described a multilevel security policy and then discussed issues such as handling polyinstantiation, using security constraints and handling the inference problem.

REFERENCES

[1] Codd E., "A Relational Model for Large Shared Data Banks", Communications of the ACM, Vol. 13, #6, 1970, pp. 377-387.

[2] Shriver B. and Wegner P., "Research Directions in Object-Oriented Programming", MIT Press, 1987.

[3] Konar A.F., Thuraisingham M.B. and Felix P., "XIMKON - An Expert Simulation and Control Program", Proceedings of the American Control Conference, Pittsburgh, PA, June 1989; also to appear in Artificial Intelligence in Process Engineering - Academic Press.

[4] Lu H., Mikkilineni K. and Thuraisingham M.B., "Design of a Distributed Data Dictionary System", Proceedings of the National Computer Conference (AFIP), Chicago, IL, June 1987.

[5] Thuraisingham M.B., "An Expert Network Simulation and Design System", Proceedings of the 7th Artificial Intelligence Conference (SPIE), Orlando, FL, March 1989.

[6] Thuraisingham M.B., "Interconnecting Heterogenous Knowledge Bases", To appear in AI-EXPERT.

[7] Thuraisingham M.B. and Larson J., "Artificial Intelligence Applications in Distributed System Design Issues", IEEE Network, Vol. 2, #6, 1988, pp. 52-60.

[8] Goldberg A. and Robson D., "Smalltalk-80, The Language and Its Implementation", Addison-Wesley, Reading, MA, 1983.

[9] Keefe T.F., Tsai W.T. and Thuraisingham M.B., "A Multilevel Security Policy for Object-Oriented Systems", Proceedings of the 11th National Computer Security Conference, Baltimore, MD, October 1988.

[10] Keefe T.F., Tsai W.T. and Thuraisingham M.B., "SODA - A Secure Object-Oriented Database System", Accepted for publication in Computers and Security.

[11] Lunt T.F. and Thuraisingham M.B., "Security for Hypermedia Systems", Unpublished Manuscript, November 21, 1988; also submitted to Computers and Security.

[12] Banerjee J. et al., "Data Model Issues for Object-Oriented Applications", ACM Transactions on Office Information Systems, Vol. 5, #1, April 1987, pp. 3-26.

[13] Thuraisingham M.B, "Mandatory Security in Object-Oriented Database Systems", Proceedings of the OOPSLA (Object-Oriented Programming: Systems, Languages and Applications) Conference (ACM) , New Orleans, LA, October 1989.

[14] Lunt T.F., "Secure Distributed Data Views Identification of Deficiencies and Directions for Future Research", A007 Final Report, Volume 4, SRI International, January 1989.

[15] Thuraisingham M.B., "Security in Object Oriented Database Systems", Accepted for publication in the Journal of Object-Oriented Programming.

[16] Thuraisingham M.B., "Design of a Multilevel Secure Object-Oriented Database System", To appear in Information Systems Journal (subject to revision).

[17] Lécluse C, Richard P. and Velez F., "O2, an Object-Oriented Data Model", Proceedings of the ACM SIGMOD Conference, Chicago, IL, June 1988.

[18] Bell D.E and LaPadula L.J., "Secure Computer Systems: Unifies Exposition and Multics Interpretation", Technical Report MTIS AD-A023588, The MITRE Corporation, July 1975.

[19] Honeywell Inc. (Dwyer, Haigh, Onuegbe, Stachour and Thuraisingham), "Secure Distributed Data Views, Implementation Specification for a Database Management System", Interim Report, RADC Contract F30602-86-C-0003, May 1988.

[20] Dwyer P., G.Jelatis and M.B.Thuraisingham, "Multilevel Security in Database Management Systems", Computers and Security, Vol. 6, #3, June 1987, pp. 252-260.

[21] Denning D.E. et al., "A Multilevel Relational Data Model", Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, CA, April 1987.

[22] Dwyer P., Onuegbe E., Stachour P. and Thuraisingham M.B., "Query Processing in LDV - A Multilevel Secure Relational Database Management System", Proceedings of the 4th Aerospace Computer Security Conference, Orlando, FL, December 1988.

[23] Stachour P., Thuraisingham M.B. and Dwyer P., "Update Processing in LDV - A Multilevel Secure Relational Database Management System", Presented at the 11th National Computer Security Conference, Baltimore, MD, October 1988.

[24] Stachour P. and Thuraisingham M.B., SQL Extensions for Security Assertions", Accepted for publication in Computer Standards and Interfaces Journal.

[25] Rougeau P. and Stearns, "The Sybase Secure Database Server", A Solution to the Multilevel Secure DBMS Problem", Proceedings of the 10th national Computer Security Conference, Baltimore, MD, October 1987.

[26] Gajnak G., "Some Results from the Entity/Relationship Multilevel Secure DBMS Project", Proceedings of the 4th Aerospace Computer Security Conference, Orlando, FL, December 1989.

27] Thuraisingham M.B., "Towards the Design of a Secure Data/Knowledge Base Management System", Accepted for publication in Data and Knowledge Engineering Journal.

[28] Thuraisingham M.B., "Recent Developments in Database Security", Tutorial Proceedings of the (IEEE) COMPSAC Conference, Orlando, FL, September 1989.

[29] Reiter R., "On Closed World Databases", in Logic and Databases, Ed: Gallaire H. and Minker J., Plenum Press, 1978.

[30] Thuraisingham M.B., "Security Checking in Relational Database Management Systems Augmented with Inference Engines", Computers and Security, Vol. 6, #6, December 1987, pp. 479 - 492.

[31] Morgenstern M., "Controlling Logical Inference in Multilevel Database Management System", Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, CA, April 1988.

[32] Thuraisingham M.B, Tsai W.T. and Keefe T.F., "Secure Query Processing using AI Techniques", Proceedings of the 21st Hawaii International Conference on Systems Sciences, January 1988.

[33] Thuraisingham M.B., "Foundations of Multilevel Databases", Presented at the 1st RADC Database Security Invitational Workshop, Menlo Park, CA, May 1988.

[34] Keefe T.F., Thuraisingham M.B. and Tsai W.T., "Secure Query Pr essing Strategies", IEEE Computer, Vol. 22, #3, March 1989, pp. 63-70.

[35] Thomsen D., Tsai W.T. and Thuraisingham M.B., "Prototyping as a Research Tool for MLS/DBMS", Proceedings of the 2nd IFIP Database Security Workshop, Kingston, Ontario, October 1988.

[36] Thuraisingham M.B., "Security Checking with Prolog Extensions", Presented at the 2nd RADC Database Security Invitational Workshop, Franconia, NH, May 1989.

[37] Zaniolo C., "Object-Oriented Programming in Prolog", Proceedings of the IEEE Logic Programming Symposium, 1984.

[38] Thuraisingham M.B., Reducibility Relationships between Decision Problems", Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, Vol. 33, 1987, pp. 305-312.

Modular Presentation of Hardware: Bounding the Reference Monitor Concept

Donald N. Dasher

30 June 1989

Abstract

Traditionally, National Computer Security Center (NCSC) evaluations have consisted of software/hardware design and implementation analysis. This analysis has focused primarily on software and paid only minimal attention to the hardware base. Recently, preliminary internal discussions have begun, exploring a more rigorous examination of the hardware.

These discussions began in response to evaluators' queries into system architecture requirements with respect to hardware. The discussions then continued in a more general vein, centering on how to evaluate hardware design in a system, if at all.

The current focus of these discussions is on gaining assurance in hardware comparable to that currently gained in software and on how evaluators will gain that assurance. It is understood that the assurance gained in hardware may be different from the assurance gained in software. In examining software, a modular presentation facilitates both the evaluator's acquisition of assurance and the analysis of the reference validation mechanism. A modular presentation not only provides the vendor and the evaluator with an excellent means of understanding the implementation, and also serves as a useful tool for bounding the reference validation mechanism. This paper will discuss why the Trusted Computer System Evaluation Criteria (TCSEC) should be interpreted to include a modular presentation of hardware within the requirements of B2 and higher levels of trust[1].

[1] The opinions expressed in this paper are those of the author and not necessarily those of his employer.

1 Introduction

One of the major quests of the evaluation team, at higher levels of trust, is to have the vendor provide adequate assurance for the system. The reference validation mechanism is the element of the system in which evaluators need to place the greatest amount of assurance. It is those parts of the system that implement isolation and access mediation that require the most detailed analysis. Isolation and access mediation occurs partially in hardware; therefore, some elements of the hardware are encompassed by the reference validation mechanism. These pieces must be evaluated to a level of detail at which the evaluator has assurance that the reference monitor is implemented properly. The best way of determining which pieces in hardware mandate this detailed investigation is to require a modular presentation of the hardware design.

2 Requirements

The TCSEC defines the reference monitor concept as

> An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects.

The reference validation mechanism is the physical implementation of the reference monitor in the system as a whole. The System Architecture requirement in the TCSEC refers directly to several types of assurances that must exist in the system. Unfortunately, commonly applied assurance techniques apply to only the software portion of the TCB. It is clear that hardware needs to have the same assurance placed on it as does software. What is not clear is what kinds of assurance apply to hardware.

Testing is the generally accepted type of hardware assurance. Yet testing alone is not enough. Without some in-depth knowledge of the hardware component under test, it is impossible to determine whether the tests provide ample coverage. In other words, in order for testing to provide assurance, the test suite has to exercise the interfaces adequately. To determine this, it is necessary to require more information on the hardware components under test than is commonly available. The problem is how to identify which components of the hardware base need a greater depth of information about their design to determine that the test suite will provide the necessary level of assurance.

3 Why A Modular Presentation is the Solution

A modular presentation of hardware will allow evaluators to determine which components in the hardware base require a closer look. The primary reason for requiring a modular presentation of the hardware base is to provide a tool for vendors and evaluators to bound the resident reference monitor. Modularized hardware is not the goal. Hardware is *modular* by design - thus requiring modularity does not bring about any more assurance. However, a modular *presentation* of hardware does add assurance. The assurance that stems from a modular presentation of hardware is a confirmation of the validity of the reference validation mechanism. A modular presentation will bound the reference monitor within the hardware. With this boundary, evaluators can determine which components need an in-depth analysis, and, based on this analysis, make a determination regarding the test suite of those components - thus bringing the necessary assurance.

Another benefit of a modularized presentation of hardware is that it will help vendors in designing and implementing their systems. Current operating systems are designed using minimal interface information. A truly trusted machine cannot be designed unless the system designer knows the hardware as well as the software. A modular presentation of the hardware will assist the vendor in better understanding the hardware base on which he is designing and thereby produce a more trusted and efficient system. In addition, the bugs associated with implementation will become easier to repair due to the programmers increased knowledge of the hardware. Finally, a modular presentation will enable the vendor to give the NCSC a significantly more assurance by providing greater insight to more of the system under evaluation.

With this, the evaluators gain assurance in not only the system, but also the vendor. The additional information allows better understanding of the system for the evaluators conducting the analysis. To summarize, the requirement of for modular presentation of hardware will provide a tool to bound the reference monitor, help vendors design and implement their system, and help evaluators gain assurance of the system's hardware - thereby contributing to the satisfaction of the TCSEC System Architecture requirement.

4 Alternatives to Modularity

Alternatives to a modular representation of hardware to bound the reference monitor concept are virtually non-existent. Evaluators could muddle through a myriad of hardware documentation trying to piece together the workings of the hardware base in order to find the elements which constitute the reference validation mechanism. This is an inefficient method of analysis requiring a standard hardware background for all evaluators. Surely, the assurance in the system design is inherently reduced when a determination of correctness is not available for the hardware base.

Another possibility is to assume that a limited set of hardware pieces will always compose the reference validation mechanism. The belief is that all you need to look at for each implementation is the address translation unit. With this philosophy, the immediate problem of what to do with unique designs and out-of-the-ordinary implementations. It is easy to imagine a hardware base that implements part of the process isolation mechanism outside of the address translation unit (i.e., the interrupt mechanism). Clearly, limiting the examination to a consistent set of components is a specific solution to a problem that demands a general approach.

5 Bounding the Reference Monitor Concept

The real problem lies in determining where in the hardware the reference validation mechanism resides. It is the responsibility of the vendor to indicate those portions of hardware which are included in the reference monitor. A modular representation of the hardware base by the vendor is the logical solution, and eases the job of both the vendor and the evaluator.

5.1 How To Modularize

Accepting this form of presentation as a requirement of the evaluation, the next issue to address is how to accomplish this modular presentation. One approach to this problem is to take a popular definition of modularity as it applies to software and modify or interpret it to fit modular presentations of hardware.

5.1.1 Software Definition

The following software definition of modularity, used in some evaluations as a guideline for software analysis, is drawn from the "Unix and B2: Are They Compatible", a paper presented at the 10th National Computer Security Conference:

> The basic assumption of the analysis was that if all modules in an operating system met the following criteria, the system could be considered fully modular. A module:
>
> - performs exactly one well-defined function
> - has well-defined parameters, interface and environment
> - interacts with other modules only in well-defined ways; and
> - is called upon to perform its function whenever that function is required.
>
> The first criterion means that a module should not combine multiple functions, particularly if they are unrelated or are also performed in other modules, and also that the results of a module should be predictable, based solely on the values of its input parameters. The second criteria means that the interface to a module should clearly reflect its implementation. The third criterion is related to the second in that parameters passed to and returned from a module should be clearly identified and have well-defined consistent meanings.

5.1.2 Definition Applied To Hardware

The above definition was created to meet the demands of software. It is not appropriate to apply it blindly to hardware, since hardware generally does not act like software (although hardware can be implemented to execute any software function and software can emulate any hardware function). The definition needs to be massaged to fit the needs of hardware (or hardware-like functions) and the intent of the proposed requirement.

- performs exactly one well-defined function;

The key word in this rule is *function*. In order to apply this to hardware, a clearer definition of what constitutes a function is desired. For example, it could be argued that processing is a function and that a CPU should therefore be an allowable module. Conversely, one might easily state that each gate array in transistor logic denotes a function because at that granularity only one function is being performed. A CPU most likely replicates multiple functions and gate arrays in other *modules*. Clearly, a middle ground must be defined. In software, functions are of equal magnitude logically. It is not certain that this is true of hardware. A more likely scenario couples this principle with a discretionary merge, allowing some functions to combine into one module based on their relative contribution toward the reference validation mechanism (i.e., a memory board could be one module). At any rate, the description of these modules must provide sufficient information to allow the evaluation team to determine which modules are part of the reference validation mechanism. Therefore, the definition of *function* has to incorporate this intent. A module in hardware must perform one well-defined collection of logic that implements a low level hardware task.

- has well-defined parameters, interface, and environment;

For software, this rule embodies the interface. In hardware, it is the interface to non-hardware-like entities which is important because this interface is exercised in an unpredictable fashion. Interaction with other hardware components is specified and predictable. Therefore, to make any judgment about the testing, the interface to software for a particular component must be analyzed.

- interacts with other modules only in well-defined ways;

This is important in determining the modules that constitute the reference validation mechanism. The interface of a module is what ultimately will be tested to derive the necessary assurance. The interaction of any given module with external components is fundamental in the bounding of the reference monitor concept. A modular presentation is essential to analyze the interaction between a module and the rest of the system.

- is called upon to perform its function whenever that function is required;

When a hardware base implements a multi-processing environment such that multiple processing modules are designed to increase speed by executing the same type of operations, this definition can become quite complicated. It is equally complicated for a system that is designed redundantly for reliability.

This rule is not appropriate for any hardware trying to achieve redundant or parallel processing. It is not yet clear how to apply this rule is a generic way.

5.2 Example

The following example presents a few generic modules that may reside in any hardware base. These components are not complete; nor are they intended to bear any resemblance to a specific hardware base. They are presented in a format similar to that a vendor might offer. For each component, the example contains a paragraph of description and a paragraph describing elements which would require a more in-depth review. It is expected that actual presentations by vendors will be more detailed for their specific hardware implementations.

5.2.1 Arithmetic Unit

> The Arithmetic Logic Unit is the unit responsible for all mathematical and logical operations that are needed by the hardware base. The unit receives an opcode and operand values from the bus. The opcode is decoded to determine what operation is to be executed. The operation is performed using a series of adders and shift registers. The resulting output is placed back onto the bus. The communications this unit uses are restricted to simple polling of the bus until the system control unit (to be described later) signals it is needed. It relies on system control to read or write memory. It executes as an isolated process independent of all other units minus the system control.

From this description the evaluation team is able to determine that this particular unit seems to have no function directly related to security or to the reference validation mechanism. There are apparently no process isolation or address translation functions performed in this module other than supporting basic arithmetic instructions. Therefore, it is expected that quality testing of the interface should provide the assurance needed due to the minimal complexity of that interface.

5.2.2 System Control Unit

> The system control unit is the unit that controls the entire system. Allocation of the bus based on process priority occurs through logic in this unit. Logic in this unit will determine which process is running and will handle process switching. The system control will handle instruction fetch and primary decode followed by distribution of the instruction to the appropriate unit for completion. The system clock is maintained here and broadcasted throughout the entire system. Other minor control functions are also the responsibility of this unit. The inputs and outputs this unit uses are primarily control signals and instruction passing. These parameters are all passed on the bus. This module controls all other components via these functions.

More information will be required on the subcomponents before a statement can be made on the adequacy of test coverage for this module because they can directly affect the enforcement of the access control policy and the TCB protection and the interface to this module is very complex and may be exercised by untrusted software. Subcomponents like the bus and its control, the process isolation logic, and instruction fetch and decode should be examined in greater detail.

5.2.3 Memory Management Unit

> The memory management unit controls access to memory. The logic within this unit accepts an instruction from the bus via the system control unit. This instruction is decoded and the memory location translated. Access is determined on a page basis. If access is allowed, the contents of the memory location are returned on the bus. If not, an error is issued. The MMU communicates with whatever unit needs data from the memory via the bus.

More information on the exact details of the memory access and the access permission check will be needed in order to analyze the adequacy of test coverage for this module because the access control policy and the TCB protection can be directly affected and of the high complexity of the interface.

5.2.4 Diagnostic and Boot Unit

> The diagnostic and boot unit is the unit that will insure the hardware base and the system are functioning correctly and in a proper state. The unit is responsible for bringing the system from a cold start into a known and predictable environment. The unit will then continuously run periodic diagnostics to check the continued correct functioning of the system. There is a bank of microcode located here that can be used to determine the cause of a failure, should one occur. The communication of this unit is with the system control unit and the interrupts and exceptions unit (to be discussed later) via control signals on the bus.

This unit will be explored further for two reasons: the System Architecture requirement and the System Integrity requirement. These two requirements combined will produce a sufficient level of detail to examine the test suite for adequacy.

5.2.5 Interrupts and Exceptions Unit

> This unit handles all of the interrupts and exceptions issued by the hardware base. The system interrupts and exceptions are prioritized to allow a uniform method in allocating the system for a given interrupt or exception. This unit primarily communicates with the system control unit, and with all other units as they issue interrupts and exceptions.

More information for this unit is needed in order to evaluate the adequacy of test coverage. It is certainly important to discover the types of interrupts and exceptions as well as how they are prioritized. It is also important to know the result of an interrupt or exception and how the system handles the switching of processes and the return of previous process context.

To reiterate, this example is a simplistic, generic sample of a modular presentation of a hardware base. It is meant only to illustrate the idea of hardware in a modular format.

6 Conclusion

As the TCSEC stands *vis a vis* the System Architecture requirement, it is apparent that there exists sufficient reasoning to dictate scrutiny of the reference monitor concept design in the entire system. The system is a combination of hardware and software. Therefore, evaluators must analyze those sections of the hardware that are within the bounds of the reference validation mechanism. In order to correctly bound the scope of reference monitor concept in hardware, the System Architecture requirement must be interpreted such that vendors are required to present their hardware base in a modular format. This format should incorporate the definition of modularity as it applies to hardware.

References

[1] *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28 - STD, Department of Defense, Washington, D.C., December 1985.

[2] Sibert, W.O, Traxler, H.M, Wagner, G.M., Downs, D.D., and Glass, J.J. *Unix and B2: Are They Compatible*, 10th National Computer Security Conference Proceedings, September 1987, pp 142-149.

Site Preparedness for the Next Network Emergency

Donald L. Alvarez
boomerC@space.mit.edu
MIT Center For Space Research

Abstract: A series of informal conversations were held to investigate local network management actions which helped or hindered recovery from the Internet Virus of November 3rd, 1988. A set of observations and recommendations are presented.

Key Words: Network, Security, Management, Computer Virus, Internet Virus, Action Plan.

In the aftermath of the Internet virus of November 3rd, 1988, the author held informal conversations with programmers and systems managers at a number of affected sites[†]. The purpose of these conversations was to look at how different sites managed their response efforts and to try to identify any common threads which the various sites found helpful in recovering from the emergency.

The following suggestions, which resulted from those conversations, are meant to provide some guidance for administrators and system managers at other sites. The suggestions are not meant to be comprehensive or even applicable to every site. Rather, they are intended to serve as a starting point for local discussions among system managers and administrators, illustrating points which some sites found to be useful or true during the course of one actual emergency. Many of the points, such as the need for off-line copies of system documentation, are almost embarrassingly obvious, yet they went unnoticed or unheeded by a large number of sites prior to the November virus.

> **(1) Resources can not be used effectively without some form of coordination. Each site should select a location with good telephone access to serve as a communications hub during an emergency.**

All sites found that some form of coordination was necessary to insure that efforts were not duplicated and to insure that separate groups did not try to recover from the virus in ways which prevented each other's success. Every site polled had instituted some form of coordination or communications hub during the emergency, but some sites did so significantly faster than others Many sites reported that either the coordinator or the coordination site moved or evolved during the course of the emergency to reflect changes in the nature of the effort under way and only rarely did the response bare any resemblance to the traditional organizational chart at the site. At many sites, instead of a formal coordinator there was only an informal message passing system. In the author's opinion, those sites which were most successful at recovering promptly and efficiently were sites where the leadership seemed to be selected based on technical expertise in a particular area rather than based on management skills or formal job title, with the leadership evolving as different technical skills were needed. Other factors may play equally significant roles in site success, however, as those sites which were most successful understandably also seemed to be blessed with an extremely high concentration of very technically competent personnel. Sites with fewer wizards and gurus may be better advised to institute a formal and static leadership hierarchy.

[†] including Harvard University, University of California at Berkeley, MIT, the Army Ballistics Research Laboratory, the Lawrence Berkeley Laboratory, and others.

A communications center for a small site should include at the very least one multi-line phone with off-site dialing capabilities and comfortable seating for a small group, possibly with a black-board or nearby conference room. A number of sites reported finding that speaker phones, conference calls, and hold buttons were the most useful technological tools during the emergency. An isolated PC with a dial-out modem is also recommended for use as a logbook and as way to download software patches from off-site.

Larger sites may also want to designate some contact point for user queries or provide a recorded message with information on the state of the system. Extremely large sites such as major universities and military bases will want to contact their public relations offices in advance to plan how to handle press inquiries in the event of another emergency. Almost every site polled reported considerable press interest in the Internet Virus, and most found this to be a significant obstacle to their recovery efforts.

(2) Information stored on-line is unlikely to be available during a network emergency. Off-line or paper documentation should be maintained for use in emergencies.

The easiest and most natural place for any system manager to store system documentation is on one of the hosts he or she manages. Documentation is most readily accessible and updateable in electronic form, and it has always been possible to load backup tapes onto another machine. When failures typically hit one or two hosts at a time, few sites found it necessary to institute a regular program of printing and storing updated copies of system documentation. With the dawn of the network virus, however, sites have become vulnerable to a new type of single-point failure. Suddenly every host at a site can be incapacitated almost simultaneously, leaving no undamaged hosts on which to read system documentation.

Network managers should institute programs to insure that accurate, up to date copies of any information needed to recover from a network emergency are maintained and stored in printed form. Regular updating of an offline set of manual pages represents one possible starting point, but ignores other relevant material, such as host tables and configuration files. Each site must decide on a case by case basis what documentation to maintain.

(3) Reconstructing the state of a network and inventing responses is extremely difficult in the face of an emergency. When possible, responses should be identified and practiced in advance.

Most large networks are in a constant state of flux. Rarely does any one system manager understand the full picture of how each host connects to every other. During an emergency, there may not be time for personnel to reconstruct the state of the network and invent appropriate responses. When likely responses can be identified, sites should work through and understand them well in advance of any real emergency.

Sample responses for which sites may wish to maintain formal written procedures include:

- isolating one or more machines from the network
- disconnecting and reconnecting local- from wide-area networks
- rebooting machines from distribution tapes or other trusted software
- halting and restarting any critical and/or real-time processes
- locating, monitoring, severing and restarting any and all network connections

In general, the larger the network, the more rapidly it changes. With a large network procedures such as these are particularly important and particularly difficult to keep up to date. To combat both of these problems, sites should institute regular "fire drills" to practice and update their network management procedures.

When compiling these lists, system managers should realize that however useful they may be during the course of an emergency, they are also exactly the information which an attacker would need to bring about the start of a local network collapse. This information should be compiled and tested regularly for accuracy, but under no circumstances should it be allowed to reside on any host which accepts either modem or off-site network connections.

(4) Phone numbers and contact iists are one of the simplest and most valuable types of system documentation. For many sites, they are also one of the least available resources.

Names and phone numbers represent one of the most vital and easily overlooked types of system documentation. Users and administrators alike need to know not only who to contact in an emergency but how to contact them. Almost every site expressed frustration that the people they most wanted to contact were known to them only by their net addresses. Network addresses were often committed to memory, but telephone numbers rarely were. Those telephone numbers which were available were generally only recorded on the disks of infected machines.

Reverse contact lists proved to be even more valuable than forward contact lists in many cases, as sites needed to know who would be likely to contact them and whether to trust them. MIT and Berkeley were particularly hampered in their efforts to work together on analyzing the virus as neither site knew whether to trust the identity of the party at the other end of the line. Several institutions reported seeing anti-viral system patches on bulletin boards but lacked the internal technical expertise to validate them, and hence did not install them because they did not know or trust the identity of the party posting the patch.

A sample starting point for contact lists to maintain and store off line includes:

- local administrators and system managers
- relevant management personnel
- network gateway managers
- vendor personnel

All of these lists should include both names and phone numbers, and should also include either home phone numbers or some type of 24 hour contact number for users if possible (the first sightings of the Internet virus occurred in the wee hours of the morning, when it was most difficult for users to find system managers). The more information, the better, but most sites indicated that lists which were incomplete but accurate and up to date were far more useful that encyclopaedic lists which were out of date or inaccurate.

Contact lists should be treated with the same care as system response plans. Many network hackers pride themselves on their "social engineering" skills. Experience has shown that attackers can and do gain system access and passwords by impersonating management or repair personnel over the telephone.

(5) Networks make up a very powerful communications medium. The decision to isolate a site from the network may in some cases be more damaging than the decision to remain connected.

Even in the face of the most serious network emergency ever, the networks themselves continued to be one of the most effective ways for distant sites to coordinate efforts. Those sites which immediately disconnected from the Internet and remained disconnected for the duration of the emergency were cut off from many sources of information which could have helped them recover from or contain the virus within their site. Sites which had access to multiple networks could make use of alternate routings and were less likely to become isolated. Bulletin boards such as provided by the USENET were of tremendous value to many sites, although in many cases the bulletin-board managers were unable to keep postings current due to related problems of their own.

Almost all parties involved praised the Internet management community for their decision to keep the mailbridges open during the emergency and allowing the flow of information to continue.

Sites with only one network connection to the outside may wish to invest in some other alternative information source, such as an account with a local bulletin board or dial-up network to insure continued access to external information.

(6) Nothing can provide absolute protection, but regular backups do protect a site against a wide variety of natural and man-made system disasters.

Sites which performed daily system backups found that they had far more latitude in choosing responses to the emergency than did sites which only performed sporadic backups. The only safe action for poorly backed-up machines was to shut down and hope, while sites with well backed-up machines could experiment, reboot, repartition disks, and even risk the possibility of reinfection, all safe under the knowledge that only a few hours or days werk could be lost at most (this was one way in which the time of the virus' release may have been an advantage. Most sites presumably backed their systems up in the evening, so that little user work would have been done between the last backup and the time of virus infection in the early morning).

With one exception, every observation or recommendation presented here centers on information -- the storage, availability, accuracy, and/or communication of information. We have fine-tuned our society for efficiency in the face of an information age. The Internet virus of November 3rd, 1988 represented the first time we had experienced even small scale information paralysis. Aside from the direct security concerns, the most important lesson that the experience has taught us is the need to prepare some level of information system to operate in the event of a catastrophic network failure, and to maintain alternate communications paths for use when our primary paths fail. These lessons will be familiar to ham radio operators, many of whom have participated in communications relays when our telephone lines were destroyed due to natural disasters. We are perhaps fortunate that we have had so little experience with large scale network emergencies. We can not expect to remain that way forever.

INTRODUCTION

INTRODUCTION

The Ethics and Education track has been added for the first time this year to accommodate the increasing demand for information on these subjects. The education, training and awareness portion of the track focuses on improving the security and privacy of sensitive information in Federal computer systems. Passage of the Computer Security Act of 1987 (Public Law 100-235) has significantly stimulated requirements in the area. The sessions cover computer security awareness training for both the employee and the executive. The ethics portion of the track addresses criminalization of computer misuse and abuse, ethics in the workplace, and the question of management responsbility versus individual rights.

The track includes refereed papers submitted in response to the Conference Call for Papers. Also included for the first time in these proceedings are Executive Summaries. These Executive Summaries highlight those presentations that were invited. Since the invited presentations are not based upon refereed papers, the Executive Summaries are intended to provide a record of their content for future reference.

It is hoped that in future years, as interest in this track broadens, more formal papers will be submitted on the topics of ethics and education. This will not only reduce the number of invited presentations, but will increase the involvement of the ethics and education communities through the formal peer review process.

Larry Martin

LARRY MARTIN
Chairman
Ethics and Education Track

EXECUTIVE SUMMARIES

Executive Summary

MAKING ELIGIBILITY FOR FEDERAL BENEFITS DETERMINATIONS UNDER THE COMPUTER MATCHING AND PRIVACY PROTECTION ACT OF 1988 (P.L. 100-503)

Robert N. Veeder
Executive Office of the President
Office of Management and Budget
Office of Information and Regulatory Affairs
Washington, DC 20503
(202) 395-4814

In a very real sense the process of governing is the process of balancing competing interests --- interests that are vying for resources, for access, for position, and the like. In designing governmental programs, especially those that deliver benefits, the planner must seek to balance two goals that are often perceived to be mutually exclusive: operational efficiency and fairness to the individuals involved. Operational efficiency is important because many programs are competing for the same resources. Inefficiencies in carrying out a program inevitably take away from what is available for other equally worthy programs. Fairness is just as important. Programs that are inherently unfair or that are operated unfairly will lose the support of those they are intended to serve. A government that is perceived to be unfair may lose the support of its citizens.

In designing systems to deliver benefits efficiently, one primary goal is to maximize the number of decisions made within the system and reduce the number made off-line. It is more efficient to treat recipients in the same way under the same processes than to attempt to adapt the system to their individual circumstances. Yet, there are times when it is important and necessary to treat individuals as individuals and not as part of a group. The efficiency goal is to make the need for such unique treatment the rare anomaly; the fairness goal is to build procedures that can accommodate such individualized determinations.

The government's use of computers to operate benefits programs helps achieve the efficient delivery of those benefits. Indeed, it would be difficult to imagine operating complex programs involving millions of people and billions of dollars without automation. Yet, surveys show that people are ambivalent about computers. While recognizing both their pervasiveness in society and their value in managing complex processes, people are concerned about many aspects of their use, especially by the government, e.g.:

- Is the information they contain accurate?
- Is it being kept safely?
- Are there ways for citizens to know what information is being kept and how it is being used?

- Are computers making determinations without any human intervention or oversight?

- Are there ways for citizens to challenge such determinations?

The Privacy Act of 1974 was one response to these concerns. The legislative history shows that Congress meant to address automated recordkeeping issues in crafting this law. Indeed, the preamble to the Act notes that Congress was concerned that the use of computers by the government could "greatly magnify the harm to individuals...." that inaccurate recordkeeping could cause. The Act attempted to involve individuals in the government's use of information about them. It gave record subjects the right to know what information the government was keeping and provided certain rights of access and amendment to those records. It also imposed responsibilities on the agencies. To help ensure compliance with its provisions, it provided civil remedies and criminal penalties.

As the Act was implemented in the 1970's, more and more of the government's information was being maintained and processed on computers. Whereas paper record data bases, because of their size and organization were difficult to use together, it became easier and easier to compare information from automated data bases. The incentives to do so rose as well. When making a determination about eligibility for a benefit that is, for example, dependent upon the amount of income and assets the applicant possesses, it is useful to have an accurate and timely way to check assets and income. By comparing automated data bases containing such information, these determinations can be made quickly.

It is at this point that the government can achieve its goal of balancing efficiency and fairness. Matching is efficient because matches can be done quickly and cheaply. It in fair because the results of such checks ensure that the scarce resources these benefits represent go only to those truly entitled to receive them. But the above is true only to the extent that the information being compared is itself complete, accurate and timely.

Congressional concern about the use of computers to make such eligibility for benefits determinations led to the first major amendment of the Privacy Act of 1974. P.L. 100-503 became law on October 18, 1988. It amended the Privacy Act to add certain protections for the subjects of Privacy Act records whose records are used in automated matching programs. These protections are essentially threefold:

- Procedural uniformity. In carrying out matching programs, Federal (and for the first time) State and local agencies are required to comply with the specific procedures set out in the Act. These include the creation of agreements defining all of the conditions under which agencies will engage in a match. The agreements are reported to Congress to permit oversight and are to be made available to the Public upon request.

- Due process for subjects. The Act gives individuals certain due process rights including advance notice that their records may be matched, notice of any adverse data found, and a chance to rebut this evidence. Before agencies can use data developed in a match to deny or suspend a benefit, they must independently verify that matching data.

- Oversight of matching. The Act establishes oversight mechanisms to ensure agency compliance. These include reports to OMB and the Congress, publication of notices in the Federal Register, and the establishment of Data

Integrity Boards at each agency engaging in matching to monitor the agency's matching activity. Data Integrity Boards play a significant role in influencing theiragency's matching activity: they must approve all matching agreements. Moreover, they serve as a repository for information about matching to help program officials make determinations about its utility.

One other significant requirement of the Act is that Data Integrity Boards evaluate matching programs in terms of their costs and benefits. This will include even programs that are mandated by statute in order to give Congress information on which to reconsider such requirements for matches that can be shown to be inefficient.

It should be noted that this Act covers a fairly narrow range of matching activities: those involving Federal benefits programs or involving substantial amounts of Federal employee personnel or financial records. It is a modest effort in that it does not cover, or specifically excludes, matching activity that has drawn criticism or concern in the past: e.g., matches fcr law enforcement or for tax enforcement. Nevertheless, it does offer significant statutory protections for what it does cover. As its provisions are implemented, it may serve as a model for future legislative initiatives, should they prove needed.

Executive Summary

PUBLIC ACCESS TO GOVERNMENT DATABASES

Anna L. Patrick
U. S. Department of Agriculture
Room 425-W, Administration Building
Washington, DC 20250

The Department of Agriculture (USDA), founded by Abraham Lincoln in 1862 to provide agricultural information to the general public, has a long history of information processing and sharing. USDA, because of the number and diversity of its programs, can serve as a good case study in addressing the issue of public access to Government databases.

It is beyond dispute that individuals and business concerns have a right to access information that has a direct bearing on their reputations, health, or well-being. It is, we believe, a responsibility of Government to provide this information at a reasonable cost, which Federal agencies have tried to do in their implementation of the Freedom of Information Act (FOIA) and in additional programs. Until now, we have provided hard copy documents in response to FOIA requests. We are now being asked to consider means of allowing electronic access to some of our databases. This is not a trivial problem nor one easily solved.

A few examples of some typical USDA programs can be used to highlight some of the problems we envision if electronic access were to be widely implemented:

> Crop Reports, on which the commodity markets--as well as the farming and agribusiness communities--depend, have, through eternal vigilance, been kept inviolate for many years. Extreme measures have been taken to assure that no individual nor business has unfair advantage from early access to Crop Reports. Our concern here is the protection of time-critical, market sensitive information.
>
> The Farmers Home Administration, which maintains a portfolio of farm loans totaling many billions of dollars, exercises great care to assure that the financial and personal details submitted with loan applications, as well as the current status of loans, are protected. Our primary concerns here are the protection of information on private citizens and the prevention of fraud committed through manipulation of financial records.
>
> We have administrative and accounting systems which process and monitor Departmental expenditures, payroll and personnel, property inventories, etc. Again, our concerns are related to personal privacy and fraud.
>
> In conducting agricultural research, some of our agencies require private industry to provide critical information regarding their products. This proprietary information is protected zealously, as it should be.

Some of the problems envisioned are not specifically security matters. For instance, agencies would have to provide additional equipment or enter into agreements with commercial time-sharing firms to provide the access required. This could add significantly to agencies' budgets at a time when economy is the watchword. If it were possible to recover costs from the public, it would still require additional Government staff to administer and operate the programs.

It has been suggested that it would be helpful to those called upon to submit information to the Government if they could submit certain data one time for multiple Government uses. It would be an ideal situation if this were possible. Unfortunately, we do not yet have standard data elements. For example, the item "name" sometimes requires last name, first name, middle initial. At other times it requires the three items in different order. And sometimes the middle name must be spelled out. We certainly support the electronic submission of information where it is possible and where the person submitting the information can be positively identified.

Our chief concern, however, is maintaining the security and privacy required by law and by common sense. The basic element of our USDA security program is establishing and maintaining individual accountability for all information processing activities. We maintain C2 security in our large centers and encourage our agencies to use the best security packages available for all their equipment, including micro-computers. It is unclear to us how members of the public could be entered into our systems as authorized users who are responsible for their actions.

The entire subject of public access to Government databases merits serious discussion and consideration. We believe that advanced technology has a role to play in our being able to achieve adequate information protection while providing this access. We should be addressing the difficult issue of standard data elements. We should be including the public in our security awareness training programs. And, in anticipation of the resolution of problems associated with public access, we should be identifying information which could be shared if controls are available.

The public has the right to access information collected and maintained on its behalf. The public has the right to expect the Government to maintain data integrity, privacy, and currency. Our task is to determine how these rights can best be guaranteed.

Executive Summary

TRENDS IN COMPUTER ABUSE/MISUSE

By JJ Buck BloomBecker
Director
National Center for Computer Crime Data
2700 N. Cahuenga Blvd.
Los Angeles, CA 90068
(213) 874-8233

In getting the attention of those one addresses, it is often useful to ask them to imagine that they have the power to change things they do not like based on the information one is about to share with them. In addressing a group as large and influential in the field of computer security as the attendees at the 12th National Computer Security Conference, it is not necessary to stretch the imagination very far to perceive the power you have.

Were there any doubt, we need only look at the program to see that amongst the panelists this morning is Mr. Joe Pujals, architect of a proposed California computer crime law, and two men whose stature has made them veritable institutions in the field of computer security, prosecutor Don Ingraham and Ernst and Whinney fellow Bill Murray.

So the focus of this briefing is a real question: what evidence would we consider if we had the power to determine what sort of laws would be effective in combatting computer crime. We do have much power in this determination, and could easily, and properly have more. I suggest that there are three trends of great significance, none of them receiving the attention it deserves.

The Sounds of Silence

As Sherlock Holmes sometimes solved mysteries by noticing things that had not occurred, I suggest that the most important characteristic trend in the area of computer abuse is what isn't happening.

Reporting Abuse

Last winter those of you who attended the 11th conference, the IEEE Security and Privacy conference, or were members of ISSA (the Information Systems Security Association) received a questionaire from the National Center for Computer Crime Data. In one of its questions, computer security practitioners (as opposed to researchers) were asked to report the number of "serious computer security incidents" of which they were aware in 1984-87, and in 1988. They were also asked to indicate how many of them were referred for prosecution.

One interpretation of the results is to put it in terms of good news and bad news. The good news was that the proportion of cases referred for prosecution tripled in 1988. The bad news is that this represented only a 6% reporting rate in 1988.[1] I invite our panelists to address the accuracy of this interpretation. Is an optimal reporting rate 100%? I'm pretty sure not. Is 6% too low? I believe so, but I can't tell you what rate would be satisfactory to me. Probably more than 50% though.

Assuming the panelists agree with my assumption that 6% is too low, I would ask their advice as to what role the computer crime laws can play in increasing the volume of reporting.

Prosecuting Abuse Criminally

The disparity in volume of prosecutions under state and federal computer crime laws is immense, and should give pause to those concerned with the use of the criminal law as a deterrent.

A survey of four jurisdictions' computerized prosecution records indicated enormous variation in the use of computer crime laws.[1] Contrast two large northeastern urban states. New York had ten prosecutions, Pennsylvania 485. California and the federal government indicated 108 prosecutions each. The federal statistic is misleading however, since it includes one case with 73 defendants.

As with the question of reporting, interpreting these statistics is much more difficult than recounting them. If we assume that more than five cases per year should be prosecuted in a major state like New York, again the question suggested is what role computer crime laws can play to increase the volume of prosecutions.

Though not exhaustive, our research shows few if any prosecutions involving computer viruses. Even if the Texas Burleson case is included, we are aware of only three alleged virus prosecutions. In view of the widespread publicity for viruses, this prosecutorial silence is troubling. I have spoken with a victim who was unable to interest law enforcement at the local, state, or federal level in prosecuting a case in which a virus was left on his machine. I have also spoken with a prosecutor who has considered prosecuting the "World Peace Virus" for some time, and so far has not done so. Both suggest that much of the activity in drafting new "anti-virus" laws ignores the problems faced in the real world of prosecution.

The fact that all but one state (Vermont) now have computer crime laws increases the need to ascertain the effectiveness of these laws.[4]

Two Cheers for Democratization

The rapid increase of access to computers continues throughout our society. In 1981, 18% of all school districts in the U.S. had at least one computer for their students. By 1987 the figure was 99%.[1] Statistics about the growth of computer use in businesses, in government, and in the home show similar, if not so dramatic increases. As a consequence, computer crime has become an "equal opportunity employer."

The Future Looks a Lot Like the Past

Our analysis of computer crime arrest figures for California suggests that much more quickly than anticipated, the profile of computer criminals is approaching that of criminals in general.[1] 32% of the computer crime arrestees in this state between 1986 and 1988 were women (68% were men). 45% of the arrestees were non-white. 34% were black, 7% hispanic, 2% other origin, and 2% unknown origin.

In a sample of prosecutions from around the country, 74% of the arrestees were not "hackers" as the term is usually understood (i.e., teenagers, usually with unusually well-developed computer skills.)[2]; Of the remainder, employees with computer access were the largest group at 26%; unemployed or criminal arrestees were next at 19%. Arrestees with computer occupations constituted 10% of the sample, ex-employees of the victims, accomplices, and law enforcement and military personnel each contributed 6% to the final total.

As a result of the "democratization" of computer crime, prosecuted cases seem far less significant than those known to the respondents to our security survey.

National Center personnel have estimated that the annual cost of computer crime in the U.S. is $555,000,000 plus 930 years of personnel time and 15.3 years of computer time.[1] This figure is an extrapolation from average losses amounting to $109,000, 365 person hours, and 26 computer hours per incident reported by our survey respondents.[1]

Reported losses in the prosecuted cases in our national survey were far less. 27.5% of the cases were in the $1,000 to $10,000 range; 20% were between $100 and $1,000; and 17.5% were between $10,000 and $100,000.[2]

The democratization of computer crime thus represents a challenge to the computer security professional. Computer crime, as defined in our criminal laws, or as defined by our prosecutorial practices, may bear insufficient resemblance to the types of computer abuse which we professionals would like to see being prosecuted. If so, I suggest that this panel may want to discuss how we can draft laws to increase the match between our concerns and our laws' protections.

Alternatives to Prosecution

The three years since the publication of the National Center for Computer Crime Data's first report, Computer Crime, Computer Security, Computer Ethics [3] have seen a significant increase in the use of non-prosecutorial strategies against computer abuse.

Civil prosecutions under the federal law have begun to occur, most notably Sprint's prosecution of a number of computerized "toll thieves", and the anti-piracy litigation of the Software Protection Association and the Business Software Association.

Administrative and organizational solutions have been discussed in connection with the "Internet virus" case. Cornell has suspended Robert Morris Jr., and members of the Association for Computing Machinery has informally discussed the question of expelling Mr. Morris.

Increasingly, computer crime laws are making provisions to increase or alter the sanctions for computer crime. The most common sanctions are seizure of arrestees computers and the court-ordered restitution to computer crime victims.[4] Holding Kevin Mitnick without bail in the federal system represents a novel and troubling sanction.

The California bill Mr. Pujals is credited with proposes two novel and controversial sanctions. As originally drafted, it would have allowed evidence of arrest for computer crime to lead to expulsion from any computer science program. A first conviction would bar the defendant from "computer-related" work in California for five years. A second conviction would extend the bar to life.

I would suggest that such sanctions should come, if at all, only after serious consultation with representatives of the computer science programs and professional organizations which they most directly would affect.

Conclusion

I used to give a speech entitled "Computer Crime, Career of the Future?" The trends I have summarized suggest that an update is appropriate. Little computer crime is reported, and what is prosecuted tends to be little computer crimes. We have yet to devise a credible and reliable set of sanctions, be they criminal law, civil law, or other organizational punishments. It is thus easy to conclude that computer crime is now the crime of the present. I challenge our panelists, and those of you in this most powerful audience who care, to address the question of what can be done to change the odds, and the public perception of the odds, against the would-be computer criminal.

References

[1] J. J. Buck BloomBecker, Commitment to Security. Los Angeles: National Center for Computer Crime Data, 1989.

[2] J. J. Buck BloomBecker, Preliminary Report. Los Angeles: National Center for Computer Crime Data, 1989.

[3] J. J. Buck BloomBecker, Computer Crime, Computer Security, Computer Ethics. Los Angeles: National Center for Computer Crime Data, 1986

[4] J. J. Buck BloomBecker, Computer Crime Law Reporter. Los Angeles: National Center for Computer Crime Data, 1988.

Executive Summary

COMPUTER ABUSE: AN ACADEMIC PERSPECTIVE

James E. Miller
Computer Science and Statistics
The University of Southern Mississippi
Hattiesburg, MS 39406 - 5106

During the time when the public was inundated with news of the exploits of teen age hackers, a commonly asked question was "Are the schools producing computer criminals?" There were numerous examples. In Atlanta, apparently motivated by computer science classes, a group of teenagers stole $250,000 worth of computer equipment, along with a significant amount of unique corporate data. A number of schools such as Carnegie-Mellon and Cornell made national news when their students were involved in a variety of computer abuses. Recent virus activity has again put the spotlight on students and schools.

In twenty plus years of working with college students, I have been fortunate not to have experienced any computer abuse incidents worthy of the evening news. Neither have I experienced any changes in the level of abuse by students that could not be explained by simply noting current enrollment levels. But computer abuse is not limited to students. The question of software piracy has clearly raised the question of faculty and administrative involvement in a much stronger way than the occasionally reported security crashing assignments that we heard about being given out in operating systems classes in the seventies and early eighties. Concerns about computer abuse in the schools have been expressed for many years.

Abuses that I have observed include: an operator who used systems privileges to seek out and then copy other students' homework for a class he was taking; a workstudy student whose job in the computer room gave him access to a transcript form which he forged to gain admittance to another university; a student with access to privileged accounts who was able to block out all other users, including the computer operator; theft of components, software, documentation, and systems; a Macintosh virus attack; numerous versions of a program for stealing passwords by mimicking the log on procedure; students borrowing, copying, stealing, and buying programs which they turned in as their own work; bogus computer generated grade sheets mailed home by a student with academic problems; the destruction or alteration of others' work; and unauthorized use of the school's computer for what apparently was a consulting venture. The only on campus computer abuse to reach the court system involved a non-enrolled student who returned to school to remove proprietary documentation which he had copied to his own file space. When his efforts were thwarted, he crashed the system.

The most commonly reported abuse is that of software piracy. I have viewed the primary offenders as faculty and administration while lab monitors have viewed it as a student problem. Our lack of success in dealing with the piracy issue is a reasonable measure of our lack of success in teaching the ethical use of computers. Recent studies have helped to clarify the magnitude of this problem.

Cohen and Cornwell [1] have reported the results of a study which both replicated and extended two earlier empirical studies (Christoph, Forcht and Bilbray 87/88 [2], and Schuster 87 [3]) that attempted to measure student attitudes toward piracy. In the Cohen/Cornwell survey, 86% felt that "...most students copy commercial software instead of buying it," as compared to 96% in the Schuster study. Agreement results of 56% and 79% were reported to a question concerning the belief that most faculty members made illegal copies of software. An additional Cohen finding, that only 25% of the surveyed students felt that administrators copied software, should probably be attributed to the students' perception of a lack of computer skills and opportunity on the part of administrators instead of one of higher ethical standards. A majority (56%) of the students who had the opportunity to pirate reported doing so.

Lin [4] conducted a survey of 100 randomly selected individuals to see if the attitudes of the general public were different from those of business faculty members surveyed in a study by Shim and Taylor [13]. Building on this study, he found generally similar responses. He did not, however, address the impact of the educational system on forming the attitudes held by the general public. Shim and Taylor had randomly selected 500 business faculty members and sent them questionnaires concerning "Unauthorized Software Copying." The results of the 218 usable questionnaires seemed to confirm student suspicions about the amount of faculty piracy activities. While 2 out of 3 faculty members felt that copying software for teaching was unethical, approximately 70% admitted doing it and 90% believed that their colleagues had.

Kim's [6] study compared and contrasted the views of computer professionals, software salespersons and teachers. He found significant differences computer professionals and sales-persons not accepting illegal software duplication while the majority of the teachers agreed that it is right to make multiple copies if used for teaching and twenty-two percent felt that "another teacher should be allowed to make a copy of their purchased course ware."

WHAT CAN A SCHOOL DO?

Businesses will point out that their employees came to them ethically flawed as products of the educational system. Colleges and universities will argue that the ethical foundation for computer and information ethics should have been established at the lower grades, and the lower grades will want to talk about the failure of families and churches. The bottom line, however, is that we now have a golden opportunity to address the problem at all levels.

It is obvious that schools need to address the problem of teaching the ethical considerations of the use of computers and the information they process. As has been pointed out by many writers and speakers, computer ethics is an area that many computer faculty are uncomfortable in lecturing on. Perhaps the findings of the Shim and Taylor quoted previously, documenting the degree of piracy by faculty, indicate that there is good justification for faculty concerns. For most teachers, computer ethics is an area where they have little or no formal training. Fortunately there is a reasonable body of literature in existence which will be of considerable value to any teacher called upon to place added emphasis on ethical considerations. The books by Donn Parker [7] and Deborah Johnson [8] are required reading, and the various computer societies can provide both information and support.

It should be evident that teachers from preschool onward must have, as an integral part of their education, some content concerning computer ethics. Each course utilizing computers should address the ethical application of the technology.

But simply teaching computer ethics will not produce our desired result. If a student is given pirated software (hotware) to use by the instructor, a piracy lecture will have little impact. Professors need to be reminded of their need to set a good ethical example for their students. Just as upper level managers must set examples for their employees, professors must do the same for their students. Implied here is the need to clearly spell out the expectations of behavior. Administrators should remember there is a need to do this for faculty also.

Where the system of presenting computer ethics appears to be weakest is in its failure to make unethical behavior unattractive. For example, when we increase the odds of detection and apply appropriate penalties for computer abuse we decrease the problem. To do otherwise is to encourage and support unethical behavior. The selection of appropriate penalities is critical to getting both faculty and student support. Shoplifting does not carry the death penalty and likewise, every computer abuse does not need to result in the permanent suspension of the students involved. What does need to happen is that the penality selected be of significant magnitude to illicit the desired behavior. The student that is caught turning in a program written by someone else and then receives a grade of zero on that assignment when that is the same grade he would have received if he had not copied has in fact been encouraged to behave unethically. An "F" grade to a student that would have failed anyway is not appropriate. The penalty needs to be something more significant such as an "F" grade and a period of restricted enrollment. And while too lenient penalities don't work, the same can be said for those that are viewed by the faculty as being too severe. In this situation one finds that abuses are ignored. When students understand that certain actions really do result in disciplinary actions they are more likely to understand that they are responsible for their own actions.

The piracy question is another one where ethical behavior can be supported and encouraged. A policy of no pirated software on school owned equipment can be enforced by not allowing pirated software to reside or be run on school owned machines. Site licenses are very attractive, because they present an inexpensive alternative to users. They also can become the best alternative if supported by good documentation, technical support when problems arise, and solid accessability.

Any successful attempt at creating an ethical environment will require faculty support. A common complaint of students is that class assignments are given that call for software that is not available. When this situation arises, the faculty member should be asked if the assignment can be modified to eliminate the problem. If there is hesitancy, the policy needs to be to place an order for the desired software. Monies for purchases under duress should come from other areas of faculty support such as travel, phones, supplies etc. This has the effect of moderating a "cookie-jar" approach to software acquisitions. The same procedure needs to be used when a faculty member (or administrator) persists in using hotware. Buy the software, with the realization that there will not be the funds for other needed activities. Faculty members also need all the support they can get in their efforts to encourage ethical behavior When disciplinary action spills out of the teacher/student relationship and enters the world of review committees, it needs to be done in a manner that does not encourage the faculty member to just ignore a similar situation.

When we look at the educational system, it is fairly easy to identify the degree to which ethics are being taught. This is of course very important, in that it is totally unreasonable to assume that individuals will instinctively know appropriate ethical decision making frameworks. But teaching ethics is only part of the solution. What we also need to look at is the degree to which we aggressively support ethical behavior and discourage that which is not.

References

[1] E. Cohen and L. Cornwell, "College Students Believe Piracy is Acceptable," CIS Education Forum, Vol. 1, Num. 3, March 1989, pp. 2-5.

[2] R. Christoph, K. Forcht and C. Bilbray, "The Development of Information Systems Ethics: An Analysis," The Journal of Computer Information Systems, Winter, 1987/1988, pp. 20-23.

[3] W. V. Schuster, "Bootlegger, Smoking Guns and Whistle Blowing: A Sad Saga of Opportunism," presentation to Western Educational Computing Conference, San Francisco, October 1987.

[4] J. Lin, "Attitudes Toward Unauthorized Software Copying: General Public vs. Business Faculty Member," SIGSMALL/PC Notes, Vol. 15, Num. 2, May 1989, pp. 3-6.

[5] J. P. Shim and G. S. Taylor "Business Faculty Members' Perceptions of Unauthorized Software Copying" OR/MS Today, 1988, pp. 30-31.

[6] D. Kim, "Moral Thinking on Computer-related Issues Among Educators and Other Related Professionals," Ph.D. Dissertation, University of Michigan, 1986.

[7] D. B. Parker, Ethical Conflicts in Computer Science and Technology, Reston, Virginia, AFIPS Press, n.d.

[8] D. G. Johnson, Computer Ethics, 2nd edition, Englewood Cliffs, New Jersey, Prentice-Hall, 1985.

Executive Summary

Access to the Access Codes '88-'89: A Prosecutor's Prospective

William J. Cook, Assistant United States Attorney

United States Attorneys Office
219 South Dearborn, Room 1500
Chicago, IL 60604
(312) 353-7602

Timely cooperation between the private sector and federal agents is essential to federal computer and telecommunication fraud prosecutions. Critical evidence can be lost if evidence of fraud is not reported in a timely manner. The purpose of this overview is to minimize the time loss by underscoring the threat and providing the framework of federal agencies involved in the enforcement effort along with an outline of federal statutes which may be used in computer and telecommunication fraud cases.

Hi-Tech Street Gangs

Some individuals cling to the notion that computer and telephone hackers are isolated Huck Finns that explore computer networks for self-education and benefit the computer industry by pushing the technology. This misguided notion is only fueled by investigators and security officers that make "wonderkid" statements about hackers to the media. These remarks only fuel hacker egos and galvanize other hackers into action.

Several months ago I observed that computer hackers were operating like hi-tech street gangs on the computer and telephone networks of this country. Nothing since then has altered my view. Many hackers now work in groups to attack access codes and computers. They are very protective about their equipment and underground networks while they take an "anything goes" approach in attacking government and corporate computers and telecommunication networks. "What's mine is mine, what's yours is debatable." When acting as a group they are capable of making and carrying out extortion demands. Many examples come from recent history.

- In June 1988, an attack was reported on the computers at the Jet Propulsion Laboratory in California.
- In October 1988, a hacker successfully broke into the personal computer of the Prime Minister of Belgium and obtained classified information.
- In October 1988, a hacker planted a virus in the New Zealand National Bank system and temporarily disabled it.
- In October 1988, Scotland Yard arrested an English attacker who had broken into over 200 military, corporate, and university computers in

the United States and Europe. The indication was that he planned to extort money from one of the victim corporations.

- In early November 1988, a Cornell undergraduate planted a computer virus that temporarily disabled 6,000 computers on the U.S. Army research computer network (ARPANET).

- In November 1988, a British hacker broke into the U.S. military computer network (MILNET) and stole non-classified government files.

- In December 1988, a search warrant filed by U.S. Customs agents in Chicago disclosed that a confederate of the Yugoslav Consul-General in Chicago was using a hacker that he set up in Dallas, Texas, to attack Dallas area defense contractors by remote access and steal computerized information. Information obtained by the Dallas hacker was subsequently smuggled out of the United States in diplomatic pouches from O'Hare Airport in Chicago with the help of the Consul-General according to the affidavit.

- In February 1989, a Chicago youth, hacker handle Shadowhawk, became the first individual tried, convicted and sentenced to prison for violating the federal Computer Fraud and Abuse Act of 1986. Shadowhawk had attacked AT&T computers at Bell Labs in Illinois, at Bell Labs in New Jersey, at a NATO missile support site in North Carolina, and at Robbins Air Force Base in Georgia. He stole copies of AT&T software worth $1.2 million and caused $174,000 worth of damage between July and September, 1987. During his trial the evidence established that Shadowhawk and other hackers methodically attacked telephone access codes enmasse on the theory that the loss would be spread out between too many people for any one person to be prosecuted.

- In March 1989, West German authorities arrested hackers and charged them with the series of computer attacks through the University of California at Berkley which were controlled and documented by Cliff Stoll. Media coverage suggested that Eastern Bloc intelligence agencies had sponsored their attacks.

- In 1989, Computerworld reported that during 1988 more than 400 computer viruses infected nearly 90,000 computers. The types of viruses jumped from 7 in February 1988 to 30 in February 1989.

- On March 9, 1989, a member of the Soviet military mission in Washington was arrested by the FBI and expelled from the United States for attempting to obtain technical information about how U.S. government computers secure classified information.

- On May 10, 1989, Kevin Mitnick, 25, plead guilty in Los Angeles to charges that he used a telephone and computer to steal a $160,000 computer security program from DEC and to possession of 16 telephone access codes. Mitnick had been held in jail as a danger to the community on charges which included allegations that he had illegally accessed NSA computers.

- On May 24, 1989, seven search warrants were executed in six states as part of a nationwide investigation of voice mail computer abuse by the U.S. Attorney's Office and the U.S. Secret Service in Chicago. Affidavits filed by the Secret Service agents described how hackers had used voice mail computers as a location for exchanging access codes and had extorted voice mail computer use from some systems operators. The affidavits noted that hackers had taken over one computer when their extortion demands were not met.

- On June 13, 1989, newspapers in Florida reported that a hacker had entered and altered Southern Bell's switching equipment to reroute calls to a probation office in Florida to a New York phone sex line.

- On June 20, 1989, Leslie Lynn Doucette a/k/a Kyrie was indicted in Chicago by a federal grand jury on wire fraud, access device fraud and computer fraud charges which alleged that she and 152 other hackers had illegally obtained $1.6 million worth of property and telecommunications services from U.S. companies through the use of access codes trafficked on voice mail computers. Court hearings in connection with the case disclosed that Doucette had been convicted in 1987 in Canada for telecommunications fraud and that she had bragged about staying 3 steps ahead of "the law." The indictment alleged that Doucette and other hackers had used voice mail boxes on voice mail computers to illegally traffic computer access codes, PBX remote access codes, telephone calling card codes and credit card information.

- On June 21, 1989 the Kansas City Star reported that a 14-year old hacker had used his computer to illegally access an Air Force satellite and confidential files of 200 companies.

The price tags on these and other computer attacks are impressive. Computer industry sources indicate that computer and telecommunication-related crime annually costs U.S. companies around $555 million. (Some estimates are as high as $5 billion.) The "gang" nature of some of these attacks by hackers are suggested in the estimate that each incident costs its victim around $450,000.00.

Tools

Congress has responded to the computer and telecommunication threat by providing federal investigators and prosecutors with impressive tools.

18 U.S.C. §1029: Prohibits fraudulent activity in connection with using access devices in interstate commerce, including computer passwords, telephone access codes and credit cards.

18 U.S.C. §1030: Prohibits remote access with intent to defraud in connection with federal interest computers and/or government-owned computers and prohibits unauthorized computer access by company employees.

18 U.S.C. §1343: Prohibits the use of interstate communications systems to further a scheme to defraud.

18 U.S.C. §2512: Prohibits making, distributing, possessing, and advertising communication interception devices and equipment.

18 U.S.C. §2314: Prohibits interstate transportation of stolen property valued at over $5,000.

17 U.S.C. §506: Prohibits copyright infringement violations - but only if the copyright is actually on file.

22 U.S.C. §2778: Prohibits illegal export of DOD-controlled software and data.

50 USCA pp 2510: Prohibits illegal export of Department of Commerce-controlled software and data.

18 U.S.C. §793: Prohibits espionage, including obtaining (and/or copying) information concerning telegraph, wireless, or signal station, building, office, research laboratory, or station for foreign government, or to injure the United States.

18 U.S.C. §2701: Prohibits unlawful access to electronically stored information.

18 U.S.C. §1362: Prohibits malicious mischief involving the willful interference with military communication systems.

18 U.S.C. §1962: Prohibits racketeering, which is in turn defined as two or more violations of specific crimes, including 18 U.S.C. §1029, 1343 and 2314.

Who Uses The Tools

The capabilities of various federal agents and agencies will vary from place to place. With that caveat, the following overview is presented:

Agency	Comments
U.S. Attorney's Office - ask for: First Assistant or Special Prosecutor	Knowledge of strengths of local federal agencies, grand jury, wire tap authority, search warrant approval, must refer espionage to DOJ.
FBI - ask for: Fraud Squad or FCI	Biggest federal law enforcement agency, international coverage, white collar fraud group copyright fraud group, FCI group, developing expertise, warrant experiences; refer to wire fraud when contacting.
Secret Service - ask for: Fraud Supervisor	Advantages of small agency, good local police contacts, statutory mandate in access device and computer fraud cases, expert HQ support, U.S. only, major cities only.

U.S. Customs - ask for: Exodus Supervisor or Exodus Coordinator	Experienced in export environment, international coverage, large agent staff, good HQ coordination with intelligence community, good coverage in major port cities and along borders, developing expertise.
Commerce Department - ask for: Enforcement Section	Small agency advantages, export environment, HQ group controls licensing, agents travel abroad to cover leads.
DCIS or DIS - ask for: Assistant SAIC	Small agency advantages, moves easily in Defense Contractor environment.

Final Observations

However, before prosecutions can be successfully brought under these sections, several things should be developed in the computer industry and the law enforcement community.

- Federal prosecutors and federal agents need to overcome "computerphobia," perhaps the leading cause of death of computer fraud cases referred to the federal government.
- Computer security specialists and systems administrators must be alert for both internal unauthorized access and external hacker attacks and the potential ramifications of such activities. They must be aware that the modern plug-in on one of their computers could be the international border in an export violation and that computerized log records may be the only evidence of espionage or "tech-theft." Unauthorized access by outside hackers and inside the company employees must be reported to law enforcement.
- Corporate and government hiring must be done carefully when the employee will have access to the computer room, computer network and/or trash from the computer room.
- Dumpster-diving is not an Olympic event, so there is no need to make your computer room trash available to the youth of America.
- Federal agents and computer security professionals must recognize the need for rapid mutual cooperation and communication, with security professionals providing background information on the attacked computer network and assisting with federal investigations and search warrant efforts.

The taxpayers and consumers that write the checks for government and private sector R&D deserve a coordinated federal law enforcement and computer industry response which recognizes that software and computer-related engineering is one of our country's greatest resources.

Executive Summary

ETHICAL USE OF COMPUTERS

Dr. Karen A. Forcht
James Madison University
Harrisonburg, VA

The subjects of computer security and computer-based crime have been the focus of substantial debate during the past decade; however, the issues involved are far from resolved. A variety of measures have been instituted, enforced, and monitored to ensure that computer centers are not vulnerable to human intervention--whether accidental or intentional. Unfortunately, this physical interpretation of security represents only one facet of a complex problem. The misuse of computer software and stored data and information may ultimately prove to be the more significant concern. In short, it is not yet clear to all parties involved in computer use just what acts should be considered as computer crime.

In the past few years, interest in the issue of ethics has been heightened as we now focus on the "people side" of computer security. The copying of a software program for a friend, while in direct violation of copyright laws, and therefore, technically a crime, may not be considered as serious to the user as stealing a physical system component or sabotaging a system for profit or revenge. The paramount question then becomes one of, "What are the definitive responsibilities of computer center employees or persons having access to software and information to the public they serve--the utltimate user or owner of information--in creating an 'environment of security' and in practicing solid ethical standards in regard to the valuable data they use when performing their jobs?"

Every culture, no matter how civilized or primitive, has an ethical code. Some codes tend to be rather formal and are entered into, unknowingly, at birth as they are a definite part of the social culture. Other ethical codes develop as we grow, becoming a vital part of our personal and professional lives. Throughout our lives, we are constantly faced with the dichotomous dilemma of right versus wrong, good versus bad.

CODES OF ETHICS

Many professional groups are attempting to formulate some definitive guidelines in this computer "sea of uncertainty" by proposing formal Codes of Ethics. The current concept today in evaluating a computer security program is "prevention on the front end--not just punishment on the back end". This "preventative maintenance" concept should be practiced by all members of the organization-- users included--to be truly effective. At the present time, there are various widely accepted Codes of Ethics in the computer profession, including:

1. British Computer Society (BCS)
   Code of Conduct
2. Data Processing Management Association (DPMA)
   Code of Ethics, Standards of Conduct and Enforcement Procedures
3. Association for Computing Machinery (ACM)
   Professional Conduct and Procedures for the Enforcement of the ACM Code of ProfessionalConduct
4. Institute of Electrical and Electronics Engineers (IEEE)
   Code of Ethics

5. Institute for Certification of Computer Professionals (ICCP)
Code of Ethics and Good Practices
6. Information Systems Security Association (ISSA)
Code of Ethics

SURVEY RESULTS

In April, 1989, two surveys were conducted at James Madison University under the auspices of the Dominion Fellowship Grant by Dr. Karen A. Forcht and Ms. Anne Myong to ascertain the level of ethical awareness and practice by college students and practitioners.

Student Survey

This survey targeted students mainly from James Madison University's College of Business and spans sophomore students through MBA's. The information was solicited from the participants by utilizing a questionnaire which included key factors such as major field of study, demographics and other personal information such as career paths, how the respondent viewed themselves and their peers morally and ethically and their personal experience with computer misuse.

The participants in the study ranged in age from 19 to 45 with a heavy concentration in the areas of Accounting, Finance, Computer Information Systems and MBA's. Most of the students were from cities ranging in population from 50,000 to 750,000+ residents. Family income was high with the heavily weighted median income being $75,000 a year or more.

Most of the students surveyed had previously had computer experience in the workplace, ranging from data entry and word processing to operations and specialized internships in the computer area. When asked if they had engaged in any form of illegal computer use, whether it be software piracy or some form of hacking, almost half of the participants admitted to using the computer for unethical means. Male hackers definitely outnumber the females and the majority of these offenders seem to be in the senior level of college and in a computer-related area of study. It is ironic and perhaps hypocritical that this same age group is adamant about their own morals and ethics which they judge to be very high.

Students who were majoring in Accounting and Computer Information Systems are the most aware of formal ethical statements and honor codes of the University than any other major. This could be attributed to the importance of accurate information produced by these two areas and the means to insure that the information is indeed correct (i.e. IRS auditors, security officers).

Alarmingly, although CIS majors and MBA candidates are aware of the ethical concerns, they are the foremost group of student hackers of all surveyed. This finding should cause great concern because these future consultants, bankers, and government officials will be working with extremely sensitive information and yet their ethical standards are lacking at this very early stage in their careers.

A comment from one of the respondent's seems to sum up the dilemma quite adequately:

"I think today more than ever, students are learning that it is more practical and safe to use the business ethics that they are taught while still in school. However, many times when the students get in a real-world situation, they may feel that they have to do certain things just to stay competitive."

Practitioner Survey

A questionnaire was mailed to the Chief Executive Officers (CEO's) of the Datamation 100 companies to ascertain their assessments concerning the ethical standards that have been formally adopted by their organizations and to seek their opinions about the ethical environment that may be present in their organization. The data analysis indicates that, for the most part, the CEO's responding adhere to a very high standard of personal ethical conduct and computer use. Furthermore, and most importantly, they expect (and require) that their employees follow ethical standards. This ethical attitude is reinforced by ethics codes, ethics awareness programs, and sanctions/reprimands of offending employees.

Some of the major survey results are:

1. When asked whether it was possible to teach ethical behavior in a classroom, rather than being learned "on the job", over 75% felt that ethics could be acquired in a classroom setting.

2. When asked whether companies should require all employees to sign an ethics oath before beginning work, over 50% agreed.

3. When asked whether companies/organizations should develop and administer an ethics awareness program for ALL employees, over 75% agreed.

4. When asked whether colleges and universities should incorporate an ethical use of computers course in their present curriculum, almost half (46.77%) agreed and 20% strongly agreed.

5. Over 80% of the respondents reported that their organizations have a formal ethics policy. Almost three-quarters (73.3%) were American companies, while only 23.3% of the foreign companies have a formal ethics policy.

6. Most of the respondents, when asked how public figures can best promote good ethics, said "by setting a good example".

CONCLUSION

These two surveys shed a great deal of light on the Ethics Awareness dilemma that is facing education and industry. Even though both groups, students and practitioners, seem to follow a very high personal standard of ethics and morals, and they obey laws, many feel that too often compromise is evident (and necessary) in the workplace in order to stay competitive.

Perhaps if educational institutions and the computer industry work together in fostering an attitude of ethical use of computers, the outcome will be a favorable, and acceptable, one. The unique and varied challenges we face in this age of information are truly unprecedented. How we achieve a balance between intellectual/ professional growth and ethical compromise--and yet remain in the "ballpark"--is indeed the paramount challenge.

Executive Summary

COMPUTER SECURITY TRAINING IN THE FEDERAL GOVERNMENT

Harold Segal
U.S. Office of Personnel Management
Office of Employee and Executive Development
P.O. Box 7559
Washington, DC 20044

Computer Security Awareness training is a significant requirement of the Computer Security Act of 1987 (Public Law 100-235). The act is augmented by computer security training regulations published by the U.S. Office of Personnel Management (OPM) and computer security training guidelines published by the National Institute of Standards and Technology (NIST). The regulations and guidelines provide Federal agencies with specific guidance on how to carry out their computer security training responsibilities.

In addition, the Office of Personnel Management has developed computer security training materials and has distributed them to all Federal agencies. The computer security awareness training materials provided to the Federal Government by the Office of Personnel Management include a video tape, instructor's guide for a one-day course, management briefing materials, desk guides, and independent study materials. The training materials are designed in a flexible format so that they may be used separately or in combination with each other. These materials have been developed in the context of the NIST guidelines and are a cost effective approach to assist agencies in fulfilling their training requirements.

The Office of Personnel Management has followed up with approximately three hundred agencies to determine how training materials are being used. Agencies are training a wide range of employees using various combinations of OPM training materials. Delivery methods vary significantly from agency to agency depending on identified training needs.

There is no single best method to carry out the training intended under Public Law 100-235. Much is left to the discretion of agency management. The presentation and discussion in this session will provide examples of what techniques are being used, what results are beginning to occur, and what does not work.

Executive Summary

SECURITY TRAINING AND AWARENESS WITHIN THE FEDERAL GOVERNMENT

Anne Todd
National Computer Security Laboratory
National Institute of Standards & Technology

The Computer Security Act of 1987, P.L. 100-235, was enacted to improve the security and privacy of sensitive information in Federal computer systems. As one way of meeting that goal, the law requires that "each agency shall provide for the mandatory periodic training in computer security awareness and accepted computer practices of all employees who are involved with the management, use, or operation of each federal computer systen within or under the supervision of that agency."

The National Institute of Standards and Technology (NIST) is responsible for developing standards, providing technical assistance, and conducting research for computers and related systems. These activities provide technical support to government and industry in the effective, safe, and economical use of computers. With the passage of P.L. 100-235, NIST's activities also include the development of standards and guidelines needed to assure the cost-effective security and privacy of information in Federal computer systems.

In fulfilling this responsibility, NIST has developed a document which provides a framework for identifying computer security training requirements for a diversity of audiences who should receive some form of computer security training. The Computer Security Training Guidelines focus on learning objectives based upon the extent to which computer security knowledge is required by an individual as it applies to his or her job function.

The Guidelines divide employees involved in the management, operation, and use of computer systems into five audience categories:

Executives	-	Senior managers responsible for setting computer security policy
Program/Functional Managers	-	Managers who have a functional responsibility for the data being processed by the computer.
IRM, Security, and Audit Personnel	-	As a group, these individuals are the competence center for protection of information resources and provide technical assistance to users, functional managers, and data processing organization in implementing agency policy on information security. They monitor its effectiveness and efficiency.

ADP Management, Operations, and Programming Staff	Implement security controls for data in their custody and advise data owners/managers of these controls. They have primary responsibility for all aspects of contingency planning.

The Guidelines provide five training content areas, or subject matter areas. The level of training required in each area will vary from general awareness training to specific courses in such areas as contingency planning, depending upon the training objectives established by the agency. The five areas are:

1. Computer Security Basics
2. Security Planning and Management
3. Computer Security Policies and Procedures
4. Contingency Planning
5. Systems Life Cycle Management.

The actual selection of the computer security training will depend upon the specific security responsibilities involving duties assigned to individual personnel. The Computer Security Training Guidelines are intended to be used by agencies as guidance in developing, acquiring, evaluating and/or selecting training courses in computer security.

In addition to the Computer Security Training Guidelines, NIST is also developing three booklets on computer security awareness. They are: Computer User's Guide to the Protection of Information Resources, Management Guide to the Protection of Information Resources, and the Executive Guide to the Protection of Information Resources.

Executive Summary

INFORMATION ETHICS, A PRACTICAL APPROACH

Harry B. DeMaio
National Manager
ProTech/Information Protection Services

Deloitte Haskins & Sells
One World Trade Center
New York, NY 10048-0601

IT'S UNNATURAL

I'd like you to consider the following assertion: Ethical behavior toward information and information resources does not come naturally to most people. An effective ethics program must take people as they are and provide guidance on how we want them to behave. The more "unnatural" that behavior seems to the individual, the more extensive and pragmatic the program must be. In this presentation, I'd like to illustrate why this lack of naturalness exists; what the implications are for information ethics, and how on a practical basis, our information ethics programs can cope with these implications.

I'm not suggesting that human beings are not fundamentally ethical. I believe that, on the whole, we are. I do mean that the rules of ethical behavior are not intuitively obvious when it comes to information. That represents a problem to managers, users and protectors of information resources. Therefore, information owners must state more explicitly and enforce more actively our ethical expectations than we usually would when dealing with the protection of tangible assets.

In fact, in many cases, the first problem is getting people to look at information as an asset at all. To most individuals, information is in an amorphous class of its own. We know for instance that "knowledge means power". However, we seldom take that statement to its logical conclusion and establish a direct asset value for that knowledge. In the first place, it's not easy to do. Secondly, somehow, it doesn't feel natural.

Therefore, I believe it's a mistake to assume that people will automatically apply their norms of ethical behavior about tangible assets to information. Further, it's unlikely that a few generic statements about behavior toward information will be sufficient, leaving it to the individual to fill in the blanks. Finally, technology has made it even more difficult for most individuals to develop, on their own, an appropriate and sharply focused information ethics code.

Why? There are probably many reasons, but I think the following four are the most basic:

1. Ethics focuses on our relations with others and their property. Information Technology can alter existing relationships and create new and unfamiliar relationships.

2. Intangible property is different and electronics has made that difference even more difficult to deal with.

3. There is a collision of rights concerning Information. Freedom of expression, freedom of information, privacy and protection of intellectual property often conflict. Sorting out priorities is difficult, especially in electronic environments.

4. There is a conflict between our natural urge to communicate and our urge to protect property.

Let's look at each of these factors individually.

1. Information Technology and Relationships

Depersonalization is probably the most obvious example. Increasingly, in the electronic environment, a system or electronic process takes the place of an interpersonal transaction. Some subconscious sense of obligation that we would feel to a human partner is reduced in the process. This is especially the case when human intervention is required but can't be supplied. Try calling for appliance service sometime. If I can't identify another person in the transaction, my sense of personal responsibility may shrink and my sense of indignation and frustration may rise.

Anonymity is another example. One of the pre-conditions that permits hackers and virus spreaders to behave as they do is the ability to hide behind some false identity. There is a corollary in the case of viruses. The victim is usually unidentified to the culprit as well. The virus attacker can ease his or her conscience by claiming they don't know what the outcome or victims will be. Therefore, they may be irresponsible but not vicious in a directed sense. I didn't say it makes sense. Selective ethics usually don't.

Therefore, electronics can weaken positive relationships and strengthen negative ones. Unless we take that into account in an ethics program, we'll miss the target.

2. Intangible Property is Different and Electronics Increases the Difference

Just look at a consolidated database and try to determine who the owner is, who the authors are, and who has rights to look, change, copy or destroy. When I distribute that same data over a large number of processors where other individuals can make alterations, additions and deletions, what's happening to the property rights of the authors and owners, whoever they may be? Unfortunately, a primary component to any property related ethics program is knowing who the owners are and what their rights are. Not easy with electronics. By the way, claiming that all information used by an enterprise is its property won't fly. Most organizations use a great deal of externally generated data which may have a specific owner or is in the public domain. Claiming ownership of everything weakens your claim on anything. For another good example of the dilemma, read some of the discussions on audio and videotape copying or software piracy.

3. Collision of Rights

Some of our guiding principles, such as freedom of information and rights of privacy, conflict in specific situations. Which is more important, my right to privacy or the public's right to know? We usually answer that question one way if we are the affected party and the other if we are the knowledge seekers. Human nature! Don't force people into double bind ethical situations by making it impossible for them to satisfy their own consciences about the behavior you expect.

4. Communicate or Keep Secret

Our natural urge is to protect private property but to share information. We also regard information as part of a general transactional relationship. You trust me. I trust you. You tell me important things. I reciprocate. That relationship is not always based on "need to know". It's more frequently based on a "want to know" and mutual accommodation. As a matter of fact, most of us feel a bit offended about the "need to know" process. Curiosity (intellectual or otherwise) is a very powerful drive. Since electronics makes it so much easier to pass information on, this mutual accommodation is made that much easier and "need to know" that much more restrictive. Further, we all have a natural suspicion of individuals or organizations who are secretive.

SO, WHAT TO DO?

1. Make the Scope Realistic

Any successful program of information ethics must take the human realities into account. We can't expect people to be perfect models of restraint without guidance, direction and management. This takes time, effort and expense. You won't get perfection under any circumstances. So, set some realistic goals and objectives and direct the program to those areas that really count. Overly ambitious ethics programs and security programs usually collapse of their own weight.

2. Make It Specific to Your Organization

Unless the individuals whose behavior you want to influence see themselves and their environment clearly in the direction you're giving, they won't respond. Philosophical statements are fine for preambles but the more localized, specific and applicable the rules are, the more likely they are to be carried out.

3. Role Play

After you create a program and before you implement it, try it out on the managers and employees who will have to live with it. Here's where the conflicts, ambiguities and hostilities will rise to the surface. They can sink a program that looks great on paper.

4. What Are You Really Saying?

Search out the implications of what you are proposing. As examples, try these two phrases:

"Need to Know". How will you determine it? How will you arbitrate? How will you enforce it? If the answers aren't clear and practical, don't use it as a principle.

"Information Owner". How determined? How arbitrated? How enforced? Again, if it can't be supported, find a different platform.

Some organizations can or must support these and other principles. Others have great difficulty. Don't pick up someone else's ethics and security program indiscriminately. Make sure it fits or it won't work. Worse yet, it may work at a cost you don't want to pay.

5. Ethical Codes Should Guide, Not Trap

You are trying to direct and guide behavior, not create snares to catch people. Yes, enforce with punitive measures but a body count is not the sign of a successful ethics program. As a matter of fact, a large number of offenders says that your ethics program is a failure.

You also have no right to demand behavior which society would regard as unreasonable unless a correspondingly strong rationale (national defense, protection of life) exists.

6. Involve Local Management Directly and Extensively

Senior management's support is important as a background. Local management will make it work. This is especially important with awareness programs. Unless an individual believes that his direct management and peers are buying in, he won't respond. A road show made up of strangers for headquarters doesn't do the trick. A local program with joint participation by the experts and local management will carry it off.

7. Peer Pressure

Th[illegible]get is to make everyone their own security officer with personal comm[illegible] and peer pressure being the most powerful motivators. The "buy-in" is tran[illegible] actions and examples, not directives and formal communications. Cre[illegible]sphere where peer pressure supports personal commitment.

8. Comm[illegible]t By Example

Finally, actions do speak louder than words. The organization, its management and its employees all demonstrate their commitment to information ethics not by the number of posters, size of the policy section or frequency of classes, but by their daily activities. That's how you should measure success and thus how you should demonstrate to your processing partners and the rest of the outside world that information ethics, natural or not, is part of your operating procedure.

Executive Summary

EXECUTIVE AWARENESS

By
Joan Forman
Bureau of Engraving and Printing
14th & C Streets, S.W.
Washington, DC 20228

By now it is obvious that a law exists requiring mandatory training for all employees involved with the management, use, or operations of Federal computers. This session offers some "friendly" techniques, advice, pit falls, and methods that were used in planning training for executives.

As the ADP Security Manager for the Bureau of Engraving and Printing (BEP), I was able to corral all the Bureau executives into a hotel room for four hours. I convinced them that it was in their best interest to listen to a stranger (contractor) speak on the foreign subject called computer security. I shall enlighten you on how this "trivial" task was accomplished.

Prior to scheduling the executive seminar, I acquired approval from the Bureau's SIRE (Senior Information Resources Executive). SIRE is the Department of the Treasury's name for the executive in charge of the AIS security program for each Bureau. Fortunately for me, the BEP SIRE is a dedicated supporter of computer security. The next step involved getting the Director, the head of the Bureau, not only to agree to the seminar, but to send a special written invitation to all Bureau executives and top managers. (Reference figure 1, the organization chart). The special invitation stated that according to the law it is mandatory for managers to attend this stimulating seminar. The Bureau calls this invitation a Special Announcement (Reference figure 2). Please take note of the next to the last sentence. That's called management incentive.

So far, so good. Now, where should this great event take place? Executives must leave the working premises. Their undivided attention is required. You need a cordial, charming, exquisite, captive, pleasant facility, all the comforts that executives are accustomed to. This was accomplished by having the Bureau's training office coordinate this endeavor. You can't lose by using their expertise. They are well trained in providing such important logistics as ensuring there are pastries, coffee, tea, juice, big soft napkins and selecting an unobjectior [illegible] facility. Plus if anything goes astray, you're not to blame. On the other hand, [illegible] ything turns out magnificent, what a grand job you performed.

Next item of concern: who has the capability of conveying this abstract foreign subject as a meaningful and interesting topic? How do you find such an individual? Again, this is where you use your training office's expertise. After numerous telephone conversations and meetings, you and your training office start interviewing the contractors. There are several important items that cannot be overlooked during this crucial time: 1) This dull and dry subject has to be made interesting to the audience, 2) It has to be made very clear that executive support is imperative in making a computer security program successful, 3) Top management must be informed of their legal responsibilities, 4) You personally must interview the instructor, 5) The subject matter must be presented without putting anyone to sleep, and 6) Above all, it's a matter of life and death as the ADP Security Manager that you have a job after the seminar has been completed. The final touch in the planning stage is to invite computer security officials from your Department. After all, the Department is actually responsible for ensuring NIST (National Institute of Standards and Technology) that they and you are abiding by the law. In addition, they may give you recognition.

Be prepared for your first joy, an adverse reaction. In my case, the Director and the SIRE were out when I received a phone call from the acting SIRE. The Deputy Director, a new kid on the block, wanted to know who said "HE" and the other top managers had to take this half-day seminar on the foreign subject of computer security. Would I please come immediately to the Deputy's office and explain this matter. After 30 minutes of explaining, the Deputy agreed to endorse the seminar, even though he did not cherish the idea. At that time, I assured him that he would truly enjoy the seminar and that he would be impressed with the knowledge he would gain. Then I left the office and had a coronary!

At this point, I had convinced the Director, Deputy Director, and the SIRE that this was the greatest thing since Mom and apple pie. What can you do to ensure that a contractor will perform the way you perceive that they should, and that you will have a job when the seminar is completed? Proceed to put the old thinking cap on and don't forget to plug it in: What was and still is the most interesting thing that has been happening in the computer security field and has been receiving national attention? BINGO! HACKING!

Now to connect hacking with computer security without losing the thrust of the topic. What else? Develop a contest! It helps a great deal if you insert a little (large amount) of humor in your seminar. After all, I am keeping your attention by using this tactic. The contest contained two awards, one for the best potential hacker and one for the individual that could best defend the Bureau from the hacker. During the seminar, making or receiving telephone calls and leaving early were announced as causes for deducting qualification points. The announcement of the contest and rules were made at the beginning of the seminar. The executives themselves selected the two winners by vote during the seminar. This is called "participation." Also, during the seminar the executives were not allowed to use their real names. They were assigned "code names" that were professionally printed (using a plotter) on cards for each executive. The prize for the best hacker was presented after a break, a copy of a hackers magazine. The prize for the defender was presented after the second break, a copy of the same hackers magazine. After all, the defender has to know what the hacker is up to. The moral is, it helps to be an inventive person to develop a gimmick that can be used to accomplish your objective.

The seminar went quite well, lots of participation from everyone including the Director, Deputy Director, and the SIRE. The Deputy Director did enjoy the seminar and mentioned it several times at other committee meetings. One Assistant Director and four top managers did not attend; however, they did attend a full-day seminar. I still have my job as the ADP Security Manager, however, after this presentation, it may be in jeopardy again. The SIRE is on the panel which is about to entertain you. Thank you for your kind attention.

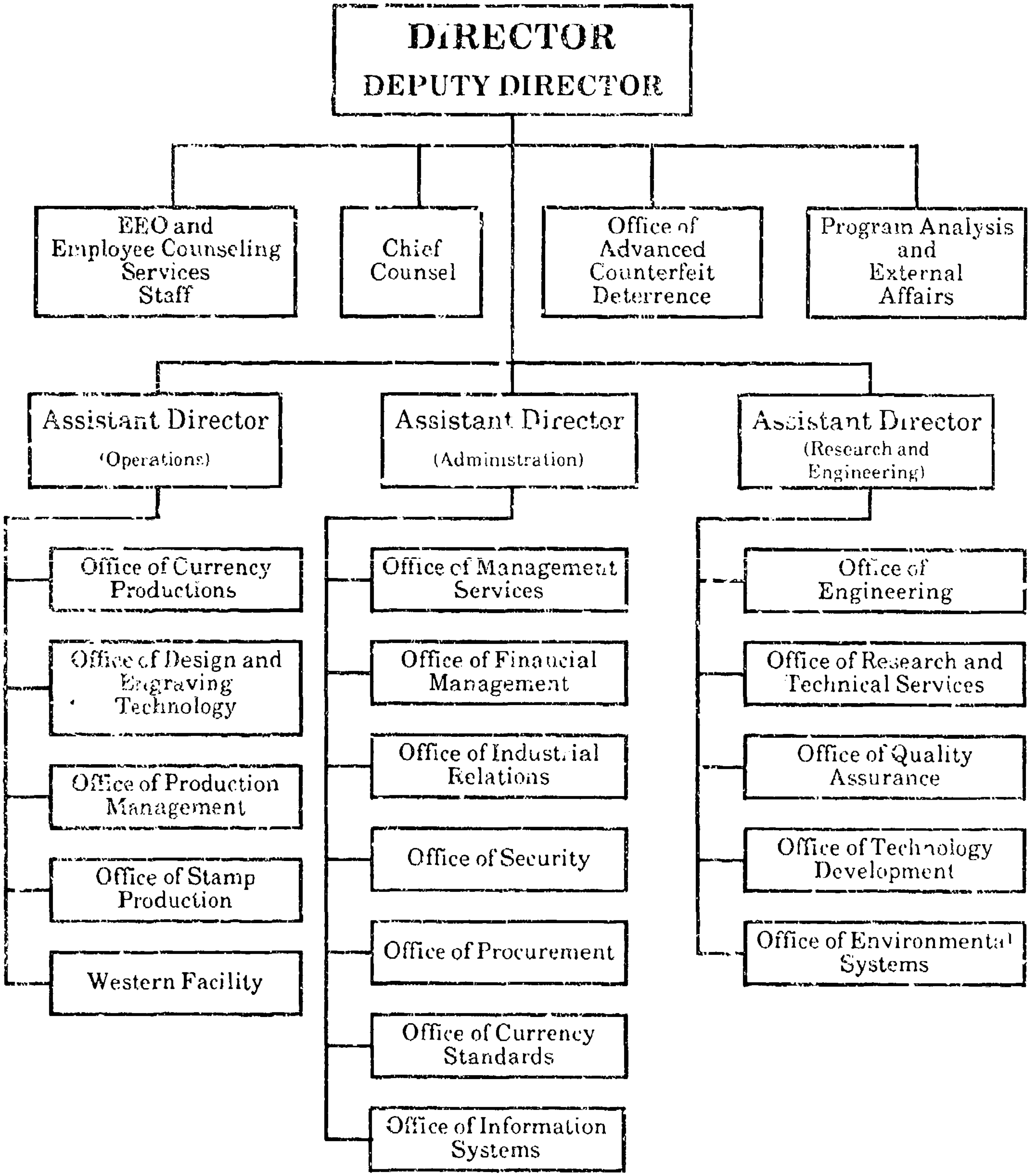

DIRECTOR
DEPUTY DIRECTOR
EEO and Employee Counseling Services Staff
Chief Counsel
Office of Advanced Counterfeit Deterrence
Program Analysis and External Affairs
Assistant Director (Operations)
Assistant Director (Administration)
Assistant Director (Research and Engineering)
Office of Currency Productions
Office of Design and Engraving Technology
Office of Production Management
Office of Stamp Production
Western Facility
Office of Management Services
Office of Financial Management
Office of Industrial Relations
Office of Security
Office of Procurement
Office of Currency Standards
Office of Information Systems
Office of Engineering
Office of Research and Technical Services
Office of Quality Assurance
Office of Technology Development
Office of Environmental Systems

FIGURE 1

CONFERENCE REFEREES

Conference Referees

Dr. Marshall D. Abrams	*The MITRE Corporation*
James P. Anderson	*James P. Anderson Company*
Alfred Arsenault	*National Computer Security Center*
Victoria Ashby	*The MITRE Corporation*
David M. Balenson	*Trusted Information Systems, Inc.*
Curt Barker	*Trusted Information Systems, Inc.*
Elaine Barker	*National Institute of Standards and Technology*
L. Kirk Barker	*Datotek*
James Barrow	*IBM*
Joseph Beckman	*National Computer Security Center*
Dr. D. Elliot Bell	*Trusted Information Systems, Inc.*
Greg Bergren	*National Computer Security Center*
Dr. Thomas A. Berson	*Anagram Laboratories*
Earl Boebert	*Secure Computing Technology Corporation*
Dr. Dennis Branstad	*National Institute of Standards and Technology*
Dr. R. Leonard Brown	*The Aerospace Corporation*
Dr. John Campbell	*National Computer Security Center*
R. O. Chester	*Martin Marietta Energy Systems*
Dr. Deborah Cooper	*Unisys Corporation*
Mark R. Cornwell	*The MITRE Corporation*
Dr. Steve Crocker	*Trusted Information Systems, Inc.*
Donald Crossman	*National Computer Security Center*
Paul F. Cudney	*Unisys Corporation*
Dr. Dorothy Denning	*Digital Equipment Corporation*
Donna Fogle Dodson	*National Institute of Standards and Technology*
Dr. Deborah Downs	*The Aerospace Corporation*
Devolyn Duggar	*National Computer Security Center*
Kenneth W. Eggers	*The MITRE Corporation*
Greg Elkmann	*National Security Agency*
Dennis Gilbert	*National Institute of Standards and Technology*
Irene Gilbert	*National Institute of Standards and Technology*
Dr. Virgil Gligor	*University of Maryland*
Harriet Goldman	*The MITRE Corporation*
Sid Green	*The MITRE Corporation*
Dr. Joshua Cuttman	*The MITRE Corporation*
Dr. Grace Hammonds	*AGCS, Inc.*
Major Douglas Hardie (USAF)	*National Computer Security Center*
Jim Healy	*COMCON*
Ronda Henning	*Harris Corp.*
Jack Holleran	*National Computer Security Center*
Jim Houser	*National Computer Security Center*
Brian Hubbard	*Trusted Information Systems, Inc.*
Howard Israel	*AT&T*
Dr. Albert Jeng	*The MITRE Corporation*
Dr. Dale M. Johnson	*The MITRE Corporation*

Thomas Keefe	*University of Minnesota*
Lisa Carnahan Kumar	*National Institute of Standards and Technology*
Leslee LaFountain	*National Computer Security Center*
Steven LaFountain	*National Computer Security Center*
Paul A. Lambert	*Motorola GEG*
Carl Landwehr	*Naval Research Laboratory*
Dr. Theodore Lee	*Trusted Information Systems, Inc.*
Nina Lewis	*University of California, Santa Barbara*
Teresa Lunt	*SRI International*
Barbara A. Mayer	*Trusted Information Systems, Inc.*
Frank Mayer	*Trusted Information Systems, Inc.*
Lynn McNulty	*National Institute of Standards and Technology*
Catherine A. Meadows	*Naval Research Laboratory*
Dr. Jonathan Millen	*The MITRE Corporation*
Andrew Moore	*Naval Research Laboratory*
Jack Moskowitz	*National Computer Security Center*
William H. Murray	*Ernst & Young*
Eugene Myers	*National Computer Security Center*
Ruth Nelson	*GTE*
Peter G. Neumann	*SRI International*
Janet Beekman Owens	*HRB Systems, Inc*
Tom Parenty	*Sybase*
Donn Parker	*SRI International*
Dr. Charles Pfleeger	*Trusted Information Systems, Inc.*
Dr. Sylvan Pinsky	*National Computer Security Center*
Phil Quade	*National Computer Security Center*
Professor Ravi Sandhu	*Ohio State University*
Marvin Schaefer	*Trusted Information Systems, Inc.*
Sam Schaen	*The MITRE Corporation*
Dr. Roger R. Schell	*Gemini Computers, Inc.*
Daniel D. Schnackenberg	*Boeing Aerospace*
Steve Schuster	*National Computer Security Center*
William R. Shockley	*Digital Equipment Corporation*
Emilie Jones Siarkiewicz	*Rome Air Development Center*
Miles Smid	*National Institute of Standards and Technology*
Brian Snow	*National Security Agency*
Dennis D. Steinauer	*National Institute of Standards and Technology*
Frank Stewart	*Anser*
Mario Tinto	*National Computer Security Center*
LTC Ray Vaughn (USA)	*National Computer Security Center*
Grant Wagner	*National Computer Security Center*
Steve Walker	*Trusted Information Systems, Inc.*
Jill Walsh	*INCO, Inc.*
Wayne Weingaertner	*National Computer Security Center*
Howard Weiss	*National Computer Security Center*
Mike White	*US House of Representatives*
Kim Wilson	*Booz-Allen & Hamilton*
Roy Wood	*National Computer Security Center*

*U.S. GOVERNMENT PRINTING OFFICE: 1989-625-529: 10519